STIC EIC 2100 Search Request Form

122966

113

Today's Date: 5/26/04

What date would you like to use to limit the search?

Priority Date: 3/20/01 Other:

Name W. Wood

AU 2124 Examiner # 79020

Room # CPK 2-5B49 Phone 305-3305

Serial # 09/812,619

Format for Search Results (Circle One):

(PAPER) DISK EMAIL

Where have you searched so far?

(USP) (DWPI) EPO JPO ACM (IBM TDB)

IEEE INSPEC SPI Other Yahoo / Google Citeseer

Is this a "Fast & Focused" Search Request? (Circle One) (YES) NO

A "Fast & Focused" Search is completed in 2-3 hours (maximum). The search must be on a very specific topic and meet certain criteria. The criteria are posted in EIC2100 and on the EIC2100 NPL Web Page at http://ptoweb/patents/stic/stic-tc2100.htm.

What is the topic, novelty, motivation, utility, or other specific details defining the desired focus of this search? Please include the concepts, synonyms, keywords, acronyms, definitions, strategies, and anything else that helps to describe the topic. Please attach a copy of the abstract, background, brief summary, pertinent claims and any citations of relevant art you have found.

Topic:

- Dynamic compiler (JAVA, Just-in-time compiler, JIT); Escape Analysis; dynamic class loading; Jalapeño; IBM

Need:

- Dynamic compiler reallocating an object from a stack (method, call, invocation) to the heap.

STIC Searcher Terese Esterhell Phone 308-7795

Date picked up 5/26/04 9:05 Date Completed

Search and Information Resources Administration

```
Set     Items   Description
S1         95   (DYNAMIC? OR RUN()TIME OR JUST()"IN"()TIME OR JIT OR JAVA)-
              ()(COMPILER? OR COMPILING) OR (RUN()TIME OR DYNAMIC()CODE)(2W-
              )GENERATION
S2    1380438   REALLOCATE? OR RETURN? OR REASSIGN? OR REALLOT? OR CHANGE(-
              )ALLOCAT? OR MOVE OR MOVING OR EARMARK?
S3    3949890   OBJECT? OR ELEMENT? OR ENTITY OR ENTITIES OR PACKET? OR IN-
             FORMATION
S4     176709   STACK?
S5    1803577   HEAP OR TEMPORARY()STORAGE OR GARBAGE()COLLECTION OR STORA-
             GE OR BUFFER? OR CACHE? OR MEMORY OR REPOSITORY? OR UMA
S6         23   ESCAPE()ANALYSIS OR DYNAMIC()CLASS()LOADING OR JALAPENO
S7          0   S1 AND S2 AND S3 AND S4
S8          0   S1 AND S2 AND S3
S9          0   S1 AND S6
S10       826   S2 AND S3 AND S4 AND S5
S11         0   S10 AND S1
S12         0   S1 AND S2 AND S4 AND S5
S13         2   S1 AND S2
S14         7   S1 AND S4
S15         6   S14 AND S5
S16      2901   S2 AND S4 AND S5
S17         0   S16 AND S1
S18         0   S16 AND S6
S19         0   S10 AND (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S20         0   S16 AND (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S21         0   S10 AND S6
S22         1   S2 AND S6
S23         5   S3 AND S6
S24         3   S4 AND S6
S25         4   S5 AND S6
S26        16   S13 OR S14 OR S15 OR S22 OR S23 OR S24 OR S25
File 347:JAPIO Nov 1976-2004/Jan(Updated 040506)
        (c) 2004 JPO & JAPIO
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200432
        (c) 2004  Thomson Derwent
```

26/5/2 (Item 2 from file: 347)
DIALOG(R)File 347:JAPIO

06636428 **Image available**
METHOD AND DEVICE FOR COMPILATION AND METHOD AND DEVICE FOR **STACK** TRACING

PUB. NO.: 2000-222242 [JP 2000222242 A]
PUBLISHED: August 11, 2000 (20000811)
INVENTOR(s): OGASAWARA TAKESHI
APPLICANT(s): INTERNATL BUSINESS MACH CORP (IBM)
APPL. NO.: 11-021942 [JP 9921942]
FILED: January 29, 1999 (19990129)
INTL CLASS: G06F-011/28; G06F-009/42; G06F-009/45; G06F-009/44

ABSTRACT
PROBLEM TO BE SOLVED: To carry out **stack** tracing even when a base pointer is omitted by storing a **storage** device with a pair of the address of a code changing a **stack** pointer and information regarding the size of a **stack** frame after the change together with a compiling process for a method.

SOLUTION: When compiling the method 40, an **JIT** **compiler** 56 generates a JITed management block 100 and registers an SP change table 102 and an initial frame size 103 in the JITed management block 100. Further, the generated method 40 is registered in a JITed code database(DB) 80. A tracer 60 traces the **stack** by using the JITed code DB80, the SP change table 102 and initial frame size 103, a frame size function 90 outputting a current frame size, and a context containing SP and a current execution address.

26/5/3 (Item 3 from file: 347)
DIALOG(R)File 347:JAPIO

06450318 **Image available**
COMPILING METHOD, METHOD AND DEVICE FOR DETECTING FRAME, CODE DESTRUCTION METHOD AND COMPUTER

PUB. NO.: 2000-035890 [JP 2000035890 A]
PUBLISHED: February 02, 2000 (20000202)
INVENTOR(s): OGASAWARA TAKESHI
APPLICANT(s): INTERNATL BUSINESS MACH CORP (IBM)
APPL. NO.: 10-267842 [JP 98267842]
FILED: September 22, 1998 (19980922)
PRIORITY: 10-132918 [JP 98132918], JP (Japan), May 15, 1998 (19980515)
INTL CLASS: G06F-009/45

ABSTRACT
PROBLEM TO BE SOLVED: To detect a frame (JITed frame) of a code in which a **memory** usable by a JIT(**Just In Time**) **compiler** of Java(R) is limited and which is compiled on a thread **stack** , in an environment where other frame coexist.

SOLUTION: When a **JIT** **compiler** becomes short of **memory** in a certain thread, all threads are once stopped and next, an active method is found in each thread, i.e., a JITed code address in a thread **stack** is found. Frames are scanned one after another from the current **stack** point SP in each thread to the bottom of the **stack** when a JITed last frame record does not exist and to the address of a JITed frame that is indicated by the latest JITed last frame record of a list when the list exists. Next, a frame that is indicated by the JITed last frame record and a JITed frame which is traced from the frame are detected.

26/5/4 (Item 4 from file: 347)
DIALOG(R)File 347:JAPIO

06386270 **Image available**
COST REDUCTION TECHNIQUE FOR DYNAMIC CLASS INITIALIZATION CHECK IN ALREADY COMPILED CODE

PUB. NO.: 11-327916 [JP 11327916 A]
PUBLISHED: November 30, 1999 (19991130)
INVENTOR(s): BAK LARS
MITROVIC SRDJAN
APPLICANT(s): SUN MICROSYST INC
APPL. NO.: 11-077718 [JP 9977718]
FILED: March 23, 1999 (19990323)
PRIORITY: 46401 [US 46401], US (United States of America), March 23, 1998 (19980323)
INTL CLASS: G06F-009/45

ABSTRACT

PROBLEM TO BE SOLVED: To provide a cost reduction technique of **dynamic class loading** and initialization check in an already compiled code.

SOLUTION: A virtual machine instruction is compiled to one or more native machine instructions even when request execution time execution **information** is not usable at the time of compilation. The native machine instruction can be provided with place holder data in a place where the request execution time execution **information** is to be present. The native machine instruction can be subscribed by the native machine instruction for transferring a control right to the section of a stab or a code for replacing the place holder data with the request execution time execution **information** at the time of execution at execution time.

26/5/5 (Item 1 from file: 350)
DIALOG(R)File 350:Derwent WPIX

015997816 **Image available**
WPI Acc No: 2004-155666/200415
XRPX Acc No: N04-124573

Java object allocating apparatus has memory **storing** dynamic compiler **which is invoked during object oriented program execution, for allocating Java object in program, to invocation** stack **frame**

Patent Assignee: INT BUSINESS MACHINES CORP (IBMC)
Inventor: SCHMIDT W J
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 20040015920	A1	20040122	US 2001812619	A	20010320	200415	B

Priority Applications (No Type Date): US 2001812619 A 20010320
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 20040015920	A1		46	G06F-009/45	

Abstract (Basic): US 20040015920 A1

NOVELTY - A **memory** coupled to a processor, stores the object oriented program. A **dynamic compiler** residing in the **memory** , is invoked during the execution of the object oriented program, by the processor. The invoked **dynamic compiler** allocates Java object in the program, to an invocation **stack** frame.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the

following:
(1) Java objects allocating method; and
(2) computer program product for allocating Java objects.
USE - For allocating Java objects in object oriented system.
ADVANTAGE - Improves performance by **dynamically compiling** portions of the code that are frequently executed.
DESCRIPTION OF DRAWING(S) - The figure shows a flowchart explaining the Java object allocation.
pp; 46 DwgNo 10/41
Title Terms: OBJECT; ALLOCATE; APPARATUS; **MEMORY** ; **STORAGE** ; DYNAMIC; COMPILE; INVOKE; OBJECT; ORIENT; PROGRAM; EXECUTE; ALLOCATE; OBJECT; PROGRAM; **STACK** ; FRAME
Derwent Class: T01
International Patent Class (Main): G06F-009/45
File Segment: EPI

26/5/6 (Item 2 from file: 350)
DIALOG(R)File 350:Derwent WPIX
(c) 2004 Thomson Derwent. All rts. reserv.

015972667 **Image available**
WPI Acc No: 2004-130508/200413
XRPX Acc No: N04-104047
Object **bound determination method for computer system, involves receiving program code in intermediate level language and analyzing program code using linear** escape analysis
Patent Assignee: MICROSOFT CORP (MICT)
Inventor: GAY D E; STEENSGAARD B
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 6681385	B1	20040120	US 99415038	A	19991007	200413	B

Priority Applications (No Type Date): US 99415038 A 19991007
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 6681385	B1		15	G06F-009/45	

Abstract (Basic): US 6681385 B1
NOVELTY - The program code is received in an intermediate level language, and analyzed with a set of rules defining a linear **escape analysis** .
DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:
(1) compiler;
(2) **storage** medium storing **object** bound determination program;
(3) method for identifying **object** bounds in program;
(4) transformation agent comprising analysis function to receive program code; and
(5) **storage** medium storing program for implementing transformation agent.
USE - For determining **object** bound in program for computer system.
ADVANTAGE - Uses linear **escape analysis** to improve **stack** allocation of **objects** , and provides a compiler for an advanced programming language.
DESCRIPTION OF DRAWING(S) - The figure shows a flowchart illustrating the **object** bound determination process.
pp; 15 DwgNo 4/6
Title Terms: **OBJECT** ; BOUND; DETERMINE; METHOD; COMPUTER; SYSTEM; RECEIVE; PROGRAM; CODE; INTERMEDIATE; LEVEL; LANGUAGE; PROGRAM; CODE; LINEAR; ESCAPE; ANALYSE
Derwent Class: T01
International Patent Class (Main): G06F-009/45
File Segment: EPI

26/5/7 (Item 3 from file: 350)
DIALOG(R)File 350:Derwent WPIX

015330287 **Image available**
WPI Acc No: 2003-391222/200337
XRPX Acc No: N03-312472
Object **oriented apparatus for allocating** objects **on an invocation** stack **used in data processing, has** object **allocation optimizer residing in** memory **and executed by processor to allocate created** object **to invocation** stack **frame**
Patent Assignee: INT BUSINESS MACHINES CORP (IBMC)
Inventor: BLAIS M N; SCHMIDT W J
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 6505344	B1	20030107	US 2000481929	A	20000112	200337	B

Priority Applications (No Type Date): US 2000481929 A 20000112
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 6505344	B1		25	G06F-009/45	

Abstract (Basic): US 6505344 B1
NOVELTY - The **object** oriented apparatus (1800) has an **object** allocation optimizer (1826) residing in a **memory** (1820) and executed by a processor (1810) to allocate the **object** , created by an instruction marked as arg escape by an **escape analysis** mechanism (1827), to an invocation **stack** frame.
DETAILED DESCRIPTION - The **escape analysis** mechanism resides in the **memory** and executed by the processor to mark each instruction in an **object** oriented program (1825) to allocate a new **object** as one of global escape, no escape and arg escape. The **object** -oriented program resides in the **memory** and comprises of several instructions. The **memory** is coupled to the processor. INDEPENDENT CLAIMS are included for the following:
(a) the **object** allocation process; and
(b) the program product of the oriented allocating apparatus.
USE - For allocating **objects** on an invocation **stack** used in data processing.
ADVANTAGE - Allows allocation of greater number of Java **objects** on invocation **stack** . Allows nested **object** to be potentially allocated on **stack** , instead of forcing each **object** to be allocated from a **heap** .
DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of the **object** -oriented apparatus.
Object oriented apparatus (1800)
Processor (1810)
Memory (1820)
Object oriented program (1825)
Object allocation optimizer (1826)
Escape analysis mechanism (1827)
pp; 25 DwgNo 18/18
Title Terms: **OBJECT** ; ORIENT; APPARATUS; ALLOCATE; **OBJECT** ; **STACK** ; DATA ; PROCESS; **OBJECT** ; ALLOCATE; OPTIMUM; **MEMORY** ; EXECUTE; PROCESSOR; ALLOCATE; **OBJECT** ; **STACK** ; FRAME
Derwent Class: T01
International Patent Class (Main): G06F-009/45
File Segment: EPI

26/5/8 (Item 4 from file: 350)
DIALOG(R)File 350:Derwent WPIX

015194319 **Image available**

WPI Acc No: 2003-254853/200325
XRPX Acc No: N03-202068
Program code performance optimizing method for Java software, involves analyzing one of byte code and intermediate language for code construct and eliminating code constructs within program code
Patent Assignee: INTEL CORP (ITLC)
Inventor: KWONG A; LAI M; WANG J
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 6484188	B1	20021119	US 99475441	A	19991230	200325	B

Priority Applications (No Type Date): US 99475441 A 19991230
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 6484188	B1		9	G06F-017/30	

Abstract (Basic): US 6484188 B1
NOVELTY - A set of interface function calls are analyzed from a native method to a Java virtual machine. Several sets of interface function calls are selected by analyzing one of a byte code and intermediate language for code constructs and eliminating or **moving** the code constructs within program code.
DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for the following:
(1) digital processing system; and
(2) computer readable medium storing program code performance optimizing program.
USE - For optimizing Java software performance in personal computer.
ADVANTAGE - Eliminates needless garbage collection operations by eliminating unnecessary structures such as if/then/else calls. Enables to be implemented in byte code accelerator.
DESCRIPTION OF DRAWING(S) - The figure shows block diagram illustrating a mixed mode **Java compiler** .
pp; 9 DwgNo 3/3
Title Terms: PROGRAM; CODE; PERFORMANCE; OPTIMUM; METHOD; SOFTWARE; ONE; BYTE; CODE; INTERMEDIATE; LANGUAGE; CODE; CONSTRUCTION; ELIMINATE; CODE; CONSTRUCTION; PROGRAM; CODE
Derwent Class: T01
International Patent Class (Main): G06F-017/30
File Segment: EPI

26/5/9 (Item 5 from file: 350)
DIALOG(R)File 350:Derwent WPIX

014575842 **Image available**
WPI Acc No: 2002-396546/200243
XRPX Acc No: N02-310999
Dynamic class loading **method in Java, involves publishing class file containing description of software** object returned **by remotely invoked** object **to calling** object
Patent Assignee: ABB RES LTD (ALLM); AUF DER MAUR D (MAUR-I); FABRI A (FABR-I); HOLLE J (HOLL-I); O'REILLY C (OREI-I)
Inventor: AUF DER MAUR D; FABRI A; HOLLE J; ORIELLY C; O'REILLY C; OREILLY C
Number of Countries: 028 Number of Patents: 004
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 1195677	A1	20020410	EP 2000810925	A	20001006	200243	B
US 20020046304	A1	20020418	US 2001968786	A	20011003	200243	
CA 2358131	A1	20020406	CA 2358131	A	20011002	200243	
NO 200104810	A	20020408	NO 20014810	A	20011003	200243	

Priority Applications (No Type Date): EP 2000810925 A 20001006

Patent Details:
Patent No Kind Lan Pg   Main IPC   Filing Notes
EP 1195677   A1 E   9 G06F-009/46
   Designated States (Regional): AL AT BE CH CY DE DK ES FI FR GB GR IE IT LI LT LU LV MC MK NL PT RO SE SI
US 20020046304 A1   G06F-009/00
CA 2358131   A1 E   G06F-009/44
NO 200104810 A   G06F-000/00

Abstract (Basic): EP 1195677 A1

NOVELTY - An **object** is **returned** by a program **object** being executable on a Java virtual machine (JVM), to a calling **object** being executable on different JVM, after being remotely invoked. A class file containing a description of the **returned** **object** , is published to the calling **object** by a publisher of a Java message system (JVM).

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for the following:

(1) **Dynamic** **class** **loading** program; and

(2) Program product containing a recorded medium storing the **dynamic** **class** **loading** program.

USE - For dynamic loading of classes in **object** oriented computing environment such as JAVA.

ADVANTAGE - The calling **object** receives the published class file, without having to know from where the class file originates. Also, since Java message system is used to publish the class file, no internet protocol addresses and internet communication unit are required for communicating with the calling **object** .

DESCRIPTION OF DRAWING(S) - The figure shows the unified modeling language notation.

pp; 9 DwgNo 1/3

Title Terms: DYNAMIC; CLASS; LOAD; METHOD; PUBLICATION; CLASS; FILE; CONTAIN; DESCRIBE; SOFTWARE; **OBJECT** ; **RETURN** ; REMOTE; INVOKE; **OBJECT** ; CALL; **OBJECT**
Derwent Class: T01
International Patent Class (Main): G06F-000/00; G06F-009/00; G06F-009/44; G06F-009/46
International Patent Class (Additional): G06F-009/40; G06F-009/54
File Segment: EPI

26/5/10 (Item 6 from file: 350)
DIALOG(R)File 350:Derwent WPIX

013482323
WPI Acc No: 2000-654266/200063
XRPX Acc No: N00-484835

Detecting long-running loops at run-time for an interpreter for selective dynamic compilation by a dynamic compiler

Patent Assignee: INT BUSINESS MACHINES CORP (IBMC)
Number of Countries: 001 Number of Patents: 001
Patent Family:
Patent No Kind Date Applicat No Kind Date Week
RD 433105 A 20000510 RD 2000433105 A 20000420 200063 B

Priority Applications (No Type Date): RD 2000433105 A 20000420
Patent Details:
Patent No Kind Lan Pg Main IPC Filing Notes
RD 433105 A 2 G06F-000/00

Abstract (Basic): RD 433105 A

NOVELTY - On first execution of a taken-backward conditional branch, the interpreter estimates the iteration count of the loop, which is used to modify the invocation counter so that compilation will be at a smaller invocation count than a threshold if the count is big enough, otherwise the method is compiled immediately. On each execution, the interpreter increments the local counter to a threshold

and updates a global counter on **return** from an interpreted method. Repeated compilation failure at the same branch is avoided by modifying the op code of a backward branch.

USE - Detecting long-running loops at run-time to improved interactive performance of virtual machines.

pp; 2 DwgNo 0/0

Title Terms: DETECT; LONG; RUN; LOOP; RUN; TIME; INTERPRETATION; SELECT; DYNAMIC; COMPILE; DYNAMIC; COMPILE

Derwent Class: T01

International Patent Class (Main): G06F-000/00

File Segment: EPI

26/5/11 (Item 7 from file: 350)

DIALOG(R)File 350:Derwent WPIX

013396441 **Image available**

WPI Acc No: 2000-568379/200053

XRPX Acc No: N00-419911

Compilation of Java program in network computing, involves storing address of execution code which changes stack **pointer and information about size of** stack **frame after change, in** memory

Patent Assignee: IBM CORP (IBMC); INT BUSINESS MACHINES CORP (IBMC)

Inventor: OGASAWARA T

Number of Countries: 002 Number of Patents: 002

Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
JP 2000222242	A	20000811	JP 9921942	A	19990129	200053	B
US 6732355	B1	20040504	US 2000493763	A	20000128	200430	

Priority Applications (No Type Date): JP 9921942 A 19990129

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
JP 2000222242	A		8	G06F-011/28	
US 6732355	B1			G06F-009/44	

Abstract (Basic): JP 2000222242 A

NOVELTY - The address of execution code which changes the **stack** pointer during compilation of a sub routine and address in which information about size of the **stack** film after change are stored, are paired and registered in change table in **memory** . Using the registered information, address which the base pointer should indicate during tracing is computed and stored.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

(a) compiler;

(b) **stack** and trace procedure; **stack** and trace system

USE - For compiling Java program in network computing.

ADVANTAGE - As the base pointer can be omitted, quality of compile code is raised and search is made easier.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of the JIT **Java compiler** in network.

pp; 8 DwgNo 5/7

Title Terms: COMPILE; PROGRAM; NETWORK; COMPUTATION; **STORAGE** ; ADDRESS; EXECUTE; CODE; CHANGE; **STACK** ; POINT; INFORMATION; SIZE; **STACK** ; FRAME; AFTER; CHANGE; **MEMORY**

Derwent Class: T01

International Patent Class (Main): G06F-009/44; G06F-011/28

International Patent Class (Additional): G06F-009/42; G06F-009/45

File Segment: EPI

26/5/12 (Item 8 from file: 350)

DIALOG(R)File 350:Derwent WPIX

013250496
WPI Acc No: 2000-422379/200036
XRAM Acc No: C00-127653
Method for flavoring ham with jalapeno **peppers by injecting ham muscle with an aqueous pickle composition, then compacting it with pretreated peppers**
Patent Assignee: HANSEL 'N GRETAL BRAND INC (HANS-N)
Inventor: ROWE J
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
US 6074682	A	20000613	US 98107927	A	19980630	200036	B

Priority Applications (No Type Date): US 98107927 A 19980630
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 6074682	A		4	A23L-001/318	

Abstract (Basic): US 6074682 A
NOVELTY - A method for making a ham product comprises:
(a) injecting ham muscle with an aqueous pickle composition; and
(b) contacting the embedded ham with **jalapeno** peppers which have been pretreated with a **buffer** .
DETAILED DESCRIPTION - An INDEPENDENT CLAIM is included for the ham product produced using the claimed method.
USE - For flavoring ham with **jalapeno** peppers.
ADVANTAGE - The nutritional content of the ham is not diminished, while the flavor is improved. The product is nutritious, convenient and low in fat.
pp; 4 DwgNo 0/0
Title Terms: METHOD; HAM; PEPPER; INJECTION; HAM; MUSCLE; AQUEOUS; PICKLE; COMPOSITION; COMPACT; PRETREATMENT; PEPPER
Derwent Class: D12
International Patent Class (Main): A23L-001/318
File Segment: CPI

26/5/13 (Item 9 from file: 350)
DIALOG(R)File 350:Derwent WPIX

013230750 **Image available**
WPI Acc No: 2000-402624/200035
XRPX Acc No: N00-301536
Memory **management in multi-threaded operating system e.g. Java virtual memory which accesses created objects anywhere other than from the thread** stack
Patent Assignee: INT BUSINESS MACHINES CORP (IBMC)
Inventor: KOLODNER E K; TROTTER M J
Number of Countries: 001 Number of Patents: 002
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
GB 2345160	A	20000628	GB 9828334	A	19981223	200035	B
GB 2345160	B	20030820	GB 9828334	A	19981223	200355	

Priority Applications (No Type Date): GB 9828334 A 19981223
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
GB 2345160	A		23	G06F-012/02	
GB 2345160	B			G06F-012/02	

Abstract (Basic): GB 2345160 A
NOVELTY - A created object in a thread **heap** (32) can be accessed anywhere other than from the thread **stack** (30) and discarded (**garbage collection**) by deleting one or more objects from the thread **heap** when **memory** space in the **heap** is required.
DETAILED DESCRIPTION - The run-time area (26) comprises the thread

memory space (28)-stores local objects,global **heap** (34)-stores shared objects,class area (36)-stores classes as they are loaded and the compiled method area (38)- used by **just - in - time compiler** to store native executable code compiled from the Java byte code.

An object is created from a class,stored,checked to see if it is global. The size of the object is created by discarding data (**garbage collection**).

INDEPENDENT CLAIMS are also included for

(a) A system for managing **memory** in a multi-threaded processing environment.

(b) A computer program product stored on a computer readable **storage** medium.

Run-time area (26)
Thread **stack** (30)
Thread **heap** (32)
Global **heap** (34)
Class area (36)
Method area (38)

USE - For maximizing available **memory** space

ADVANTAGE - Can be used on any platform not necessarily Pentium

DESCRIPTION OF DRAWING(S) - The figure shows a schematic of a Java virtual machine embodying components involved in **memory** management.

pp; 23 DwgNo 2/6

Title Terms: **MEMORY** ; MANAGEMENT; MULTI; THREAD; OPERATE; SYSTEM; VIRTUAL; **MEMORY** ; ACCESS; OBJECT; THREAD; **STACK**
Derwent Class: T01
International Patent Class (Main): G06F-012/02
File Segment: EPI

26/5/14 (Item 10 from file: 350)
DIALOG(R)File 350:Derwent WPIX

012796585 **Image available**
WPI Acc No: 1999-602815/199952
XRPX Acc No: N99-444540

Virtual machine for executing a virtual machine instruction sequence under control of a real machine

Patent Assignee: MATSUSHITA ELECTRIC IND CO LTD (MATU); MATSUSHITA DENKI SANGYO KK (MATU); FUJITA M (FUJI-I); HAYAMA S (HAYA-I); INOUE S (INOU-I); ISHIKAWA A (ISHI-I); WAKI H (WAKI-I)
Inventor: FUJITA M; HAYAMA S; INOUE S; ISHIKAWA A; WAKI H
Number of Countries: 027 Number of Patents: 003
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 949564	A2	19991013	EP 99302715	A	19990407	199952	B
JP 11296381	A	19991029	JP 9896204	A	19980408	200003	
US 20030233386	A1	20031218	US 99288263	A	19990408	200401	
			US 2003403917	A	20030331		

Priority Applications (No Type Date): JP 9896204 A 19980408
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 949564	A2	E	136	G06F-009/00	
	Designated States (Regional): AL AT BE CH CY DE DK ES FI FR GB GR IE IT LI LT LU LV MC MK NL PT RO SE SI				
JP 11296381	A		61	G06F-009/455	
US 20030233386	A1			G06F-009/00	Div ex application US 99288263

Abstract (Basic): EP 949564 A2

NOVELTY - The virtual machine includes **stack** (120) for temporarily storing data. An instruction store (102) holds the virtual machine instruction sequence and several sets of succeeding instruction information. Each instruction is associated with a set of succeeding instruction information that indicates a change in a **storage** state of the data in the **stack** due to execution of an instruction executed

after the associated instruction. A read circuit reads an instruction and an associated set of succeeding instruction information from the instruction store decoding-executor (103,110) to specify and execute operations corresponding to a combination of the read virtual machine instruction and the read set of succeeding instruction information.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also given for:

(a) a compiler that generates programs for a virtual machine;

(b) a Just-In-Time (**JIT**) **compiler** for use with a virtual machine that executes a virtual machine instruction sequence under control of a real machine;

(c) a **storage** method used by an instruction store that holds a virtual machine instruction sequence to be executed by a virtual machine; and

(d) a computer-readable recording medium that stores a program to have a computer function as a virtual machine with a **stack** architecture.

USE - For use within the field of communications, in a network with different kinds of CPU (real machine).

ADVANTAGE - Increases the execution speed of a virtual machine. To do this, it has diminished the number of true data dependencies, increased the quality of interrupt handling and uses a high speed compiler so that it does not have to deal with complex address handling.

DESCRIPTION OF DRAWING(S) - The drawing shows a block diagram of the construction of the virtual machine.

Instruction store (102)

Instruction store decoding-executor (103,110)

Virtual machine includes **stack** (120)

pp; 136 DwgNo 35/103

Title Terms: VIRTUAL; MACHINE; EXECUTE; VIRTUAL; MACHINE; INSTRUCTION; SEQUENCE; CONTROL; REAL; MACHINE

Derwent Class: T01

International Patent Class (Main): G06F-009/00; G06F-009/455

File Segment: EPI

26/5/15 (Item 11 from file: 350)

012744665 **Image available**

WPI Acc No: 1999-550782/199946

XRPX Acc No: N99-407563

Software development tool, especially compiler, and Java programming language, providing machine independent solution suitable for high performance systems development

Patent Assignee: CYGNUS SOLUTIONS (CYGN-N)

Inventor: BOTHNER P

Number of Countries: 084 Number of Patents: 005

Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
WO 9942925	A1	19990826	WO 99US3518	A	19990218	199946	B
AU 9927727	A	19990906	AU 9927727	A	19990218	200003	
US 6110226	A	20000829	US 9825842	A	19980219	200043	
EP 1062575	A1	20001227	EP 99908246	A	19990218	200102	
			WO 99US3518	A	19990218		
JP 2002504725	W	20020212	WO 99US3518	A	19990218	200215	
			JP 2000532796	A	19990218		

Priority Applications (No Type Date): US 9825842 A 19980219

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
WO 9942925	A1	E	28	G06F-009/45	

Designated States (National): AL AM AT AU AZ BA BB BG BR BY CA CH CN CU CZ DE DK EE ES FI GB GD GE GH GM HR HU ID IL IN IS JP KE KG KP KR KZ LC LK LR LS LT LU LV MD MG MK MN MW MX NO NZ PL PT RO RU SD SE SG SI SK SL TJ TM TR TT UA UG US UZ VN YU ZW

Designated States (Regional): AT BE CH CY DE DK EA ES FI FR GB GH GM GR IE IT KE LS LU MC MW NL OA PT SD SE SZ UG ZW

AU 9927727	A			Based on patent WO 9942925
US 6110226	A		G06F-009/45	
EP 1062575	A1 E		G06F-009/45	Based on patent WO 9942925
	Designated States (Regional): DE FR GB SE			
JP 2002504725 W		29	G06F-009/45	Based on patent WO 9942925

Abstract (Basic): WO 9942925 A1

NOVELTY - Coherent model allows precompiled Java code, interpreted byte code, compiled Java code, and C/C++ code to interoperate. Optimizing ahead-of-time **Java compiler** compiles either Java source code (517) or byte code (519). Code distributed only in byte code may be precompiled. Java **stack** slot compilation scheme achieves code optimization and overcomes difficulty peculiar to ahead-of-time Java compilation.

USE - For providing a Java development environment using optimizing ahead-of-time compiler.

ADVANTAGE - A static layout of Java metadata is created by the compiler, obviating the need to create such a layout at runtime.

DESCRIPTION OF DRAWING(S) - The drawing shows a block diagram of a Java implementation model in accordance with one aspect of the tool.

source code to machine code compiler (517)

byte code to machine code compiler (519)

pp; 28 DwgNo 5/11

Title Terms: SOFTWARE; DEVELOP; TOOL; COMPILE; PROGRAM; LANGUAGE; MACHINE; INDEPENDENT; SOLUTION; SUIT; HIGH; PERFORMANCE; SYSTEM; DEVELOP

Derwent Class: T01

International Patent Class (Main): G06F-009/45

File Segment: EPI

26/5/16 (Item 12 from file: 350)

DIALOG(R)File 350:Derwent WPIX

009122345

WPI Acc No: 1992-249782/199230

Related WPI Acc No: 1992-249787

XRAM Acc No: C92-111418

Dehydrated vegetable prods. prepn. contg. solid amorphous osmotic agent - gives extended storage **life and good flavour, texture, colour and taste on rehydration**

Patent Assignee: MCCORMICK & CO INC (MCCO-N)

Inventor: AEBI K J; AUNG T; GRYPA R D; LEE M J; FULGER C V

Number of Countries: 040 Number of Patents: 009

Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
WO 9210940	A1	19920709	WO 91US9197	A	19911224	199230	B
AU 9191681	A	19920722	AU 9191681	A	19911224	199244	
			WO 91US9197	A	19911224		
JP 5015303	A	19930126	JP 91341563	A	19911224	199309	
EP 564573	A1	19931013	WO 91US9200	A	19911224	199341	
			EP 92903501	A	19911224		
TW 210279	A	19930801	TW 91110034	A	19911223	199345	
TW 224422	A	19940601	TW 91110036	A	19911223	199426	
US 5368873	A	19941129	US 90630967	A	19901224	199502	
			US 91799145	A	19911127		
			US 93110680	A	19930809		
IL 100200	A	19950124	IL 100200	A	19911129	199510	
JP 8116866	A	19960514	JP 91361258	A	19911224	199629	

Priority Applications (No Type Date): US 91799145 A 19911127; US 90630967 A 19901224; US 91735954 A 19910725; US 93110680 A 19930809

Cited Patents: GB 946330; US 2432222; US 3556814; US 3801714; US 4361589; US 4447460; US 4777055; US 4832969; US 4889730; US 4948609

Patent Details:

```
Set     Items   Description
S1        746   (DYNAMIC? OR RUN()TIME OR JUST()"IN"()TIME OR JIT OR JAVA)-
              ()(COMPILER? OR COMPILING) OR (RUN()TIME OR DYNAMIC()CODE)(2W-
              )GENERATION
S2     438234   REALLOCATE? OR RETURN? OR REASSIGN? OR REALLOT? OR CHANGE(-
              )ALLOCAT? OR MOVE OR MOVING OR EARMARK?
S3    5017161   OBJECT? OR ELEMENT? OR ENTITY OR ENTITIES OR PACKET? OR IN-
              FORMATION
S4     114433   STACK?
S5     908914   HEAP OR TEMPORARY()STORAGE OR GARBAGE()COLLECTION OR STORA-
              GE OR BUFFER? OR CACHE? OR MEMORY OR REPOSITORY? OR UMA
S6        223   ESCAPE()ANALYSIS OR DYNAMIC()CLASS()LOADING OR JALAPENO
S7          0   S1 AND S2 AND S3 AND S4
S8          9   S1 AND S2 AND S3
S9         18   S1 AND S6
S10        96   S2 AND S3 AND S4 AND S5
S11         0   S10 AND S1
S12         0   S1 AND S2 AND S4 AND S5
S13        16   S1 AND S2
S14        54   S1 AND S4
S15        22   S14 AND S5
S16       333   S2 AND S4 AND S5
S17         0   S16 AND S1
S18         1   S16 AND S6
S19         1   S10 AND (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S20         5   S16 AND (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S21         1   S10 AND S6
S22         7   S2 AND S6
S23       117   S3 AND S6
S24        26   S4 AND S6
S25        73   S5 AND S6
S26        18   S24 AND S25
S27        23   S23 AND S24
S28        74   S8 OR S13 OR S15 OR S18 OR S19 OR S20 OR S21 OR S22 OR S24
              OR S26 OR S27
S29        48   S28 NOT PY>2001
S30        48   S29 NOT PD>20010320
S31        39   RD (unique items)
File   8:Ei Compendex(R) 1970-2004/May W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2004/Apr
         (c) 2004 ProQuest Info&Learning
File 202:Info. Sci. & Tech. Abs. 1966-2004/May 14
         (c) 2004 EBSCO Publishing
File  65:Inside Conferences 1993-2004/May W4
         (c) 2004 BLDSC all rts. reserv.
File   2:INSPEC 1969-2004/May W3
         (c) 2004 Institution of Electrical Engineers
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File  94:JICST-EPlus 1985-2004/May W1
         (c)2004 Japan Science and Tech Corp(JST)
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Apr
         (c) 2004 The HW Wilson Co.
File  95:TEME-Technology & Management 1989-2004/May W2
         (c) 2004 FIZ TECHNIK
```

31/5/1 (Item 1 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05927964 E.I. No: EIP01436705800
Title: Incrementalized pointer and escape analysis
Author: Vivien, F.; Rinard, M.
Corporate Source: ICPS/LSIIT Universite Louis Pasteur, Strasbourg, France
Conference Title: ACM SIGPLAN'01 Conference on Programming Language Design and Implementation (PLDI)
Conference Location: Snowbird, UT, United States Conference Date: 20010620-20010622
Sponsor: ACM SIGPLAN
E.I. Conference No.: 58601
Source: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) 2001. p 35-46
Publication Year: 2001
CODEN: PSPIEM
Language: English
Document Type: CA; (Conference Article) Treatment: T; (Theoretical); X; (Experimental)
Journal Announcement: 0111W1
Abstract: We present a new pointer and **escape analysis** . Instead of analyzing the whole program, the algorithm incrementally analyzes only those parts of the program that may deliver useful results. An analysis policy monitors the analysis results to direct the incremental investment of analysis resources to those parts of the program that offer the highest expected optimization **return** . Our experimental results show that almost all of the objects are allocated at a small number of allocation sites and that an incremental analysis of a small region of the program surrounding each site can deliver almost all of the benefit of a whole-program analysis. Our analysis policy is usually able to deliver this benefit at a fraction of the whole-program analysis cost. 17 Refs.
Descriptors: *Computer programming; Software engineering; Query languages ; Synchronization; Optimization; Graph theory; Algorithms
Identifiers: Pointer and **escape analysis**
Classification Codes:
723.1.1 (Computer Programming Languages)
723.1 (Computer Programming); 731.1 (Control Systems); 921.5 (Optimization Techniques); 921.4 (Combinatorial Mathematics, Includes Graph Theory, Set Theory)
723 (Computer Software, Data Handling & Applications); 731 (Automatic Control Principles & Applications); 921 (Applied Mathematics)
72 (COMPUTERS & DATA PROCESSING); 73 (CONTROL ENGINEERING); 92 (ENGINEERING MATHEMATICS)

31/5/2 (Item 2 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05818295 E.I. No: EIP01216510918
Title: Java runtime systems: Characterization and architectural implications
Author: Radhakrishnan, R.; Vijaykrishnan, N.; John, L.K.; Sivasubramaniam, A.; Rubio, J.; Sabarinathan, J.
Corporate Source: Laboratory for Computer Architecture Dept. of Elec. and Computer Eng. University of Texas at Austin, Austin, TX 78712, United States
Source: IEEE Transactions on Computers v 50 n 2 February 2001 2001. p 131-146
Publication Year: 2001
CODEN: ITCOB4 ISSN: 0018-9340
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 0105W4
Abstract: The Java Virtual Machine (JVM) is the cornerstone of Java

technology and its efficiency in executing the portable Java bytecodes is crucial for the success of this technology. Interpretation, Just-In-Time (JIT) compilation, and hardware realization are well-known solutions for a JVM and previous research has proposed optimizations for each of these techniques. However, each technique has its pros and cons and may not be uniformly attractive for all hardware platforms. Instead, an understanding of the architectural implications of JVM implementations with real applications can be crucial to the development of enabling technologies for efficient Java runtime system development on a wide range of platforms. Toward this goal, this paper examines architectural issues from both the hardware and JVM implementation perspectives. The paper starts by identifying the important execution characteristics of Java applications from a bytecode perspective. It then explores the potential of a smart **JIT compiler** strategy that can dynamically interpret or compile based on associated costs and investigates the CPU and **cache** architectural support that would benefit JVM implementations. We also study the available parallelism during the different execution modes using applications from the SPECjvm98 benchmarks. At the bytecode level, it is observed that less than 45 out of the 256 bytecodes constitute 90 percent of the dynamic bytecode stream. Method sizes fall into a trinodal distribution with peaks of 1, 9, and 26 bytecodes across all benchmarks. The architectural issues explored in this study show that, when Java applications are executed with a **JIT compiler** , selective translation using good heuristics can improve performance, but the saving is only 10-15 percent at best. The instruction and data **cache** performance of Java applications are seen to be better than that of C/C++ applications except in the case of data **cache** performance in the JIT mode. Write misses resulting from installation of **JIT compiler** output dominate the misses and deteriorate the data **cache** performance in JIT mode. A study on the available parallelism shows that Java programs executed using **JIT compilers** have parallelism comparable to C/C++ programs for small window sizes, but falls behind when the window size is increased. Java programs executed using the interpreter have very little parallelism due to the **stack** nature of the JVM instruction set, which is dominant in the interpreted execution mode. In addition, this work gives revealing insights and architectural proposals for designing an efficient Java runtime system. 46 Refs.

Descriptors: Java programming language; Response time (computer systems); Computer architecture; Parallel processing systems; Just in time production ; Program compilers; Optimization; **Cache memory** ; C (programming language); Benchmarking

Identifiers: Java virtual machine; Java runtime systems; Central processing unit; Data **cache** ; Java bytecodes

Classification Codes:

723.1.1 (Computer Programming Languages)

723.1 (Computer Programming); 722.4 (Digital Computers & Systems); 913.1 (Production Engineering); 921.5 (Optimization Techniques); 722.1 (Data Storage, Equipment & Techniques)

723 (Computer Software, Data Handling & Applications); 722 (Computer Hardware); 913 (Production Planning & Control; Manufacturing); 921 (Applied Mathematics)

72 (COMPUTERS & DATA PROCESSING); 91 (ENGINEERING MANAGEMENT); 92 (ENGINEERING MATHEMATICS)

31/5/3 (Item 3 from file: 8)

05804686 E.I. No: EIP00025069935

Title: Compositional pointer and escape analysis **for Java programs**

Author: Whaley, John; Rinard, Martin

Corporate Source: Massachusetts Inst of Technology, Cambridge, MA, USA

Conference Title: Proceedings of the 1999 Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA-99

Conference Location: Denver, CO, USA Conference Date: 20991101-20991105

E.I. Conference No.: 56092

Source: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA v 34 n 10 Oct 1999. ACM, New York, NY, USA. p 187-206
Publication Year: 1999
CODEN: 002376
Language: English
Document Type: CA; (Conference Article) Treatment: G; (General Review)
Journal Announcement: 0104W3
Abstract: This paper presents a combined pointer and **escape analysis** algorithm for Java programs. The algorithm is based on the abstraction of points-to escape graphs, which characterize how local variables and fields in **objects** refer to other **objects** . Each points-to escape graph also contains escape **information** , which characterizes how **objects** allocated in one region of the program can escape to be accessed by another region. The algorithm is designed to analyze arbitrary regions of complete or incomplete programs, obtaining complete **information** for **objects** that do not escape the analyzed regions. We have developed an implementation that uses the escape **information** to eliminate synchronization for **objects** that are accessed by only one thread and to allocate **objects** on the **stack** instead of in the **heap** . Our experimental results are encouraging. We were able to analyze programs tens of thousands of lines long. For our benchmark programs, our algorithms enable the elimination of between 24% and 67% of the synchronization operations. They also enable the **stack** allocation of between 22% and 95% of the **objects** . (Author abstract) 29 Refs.
Descriptors: Java programming language; Algorithms; Data flow analysis; Synchronization; Benchmarking; Resource allocation; **Object** oriented programming
Identifiers: **Escape analysis** algorithm; Points to escape graphs; Benchmark programs
Classification Codes:
723.1.1 (Computer Programming Languages)
723.1 (Computer Programming); 723.5 (Computer Applications); 912.2 (Management)
723 (Computer Software); 912 (Industrial Engineering & Management)
72 (COMPUTERS & DATA PROCESSING); 91 (ENGINEERING MANAGEMENT)

31/5/4 (Item 4 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05804675 E.I. No: EIP00025069924
Title: Escape analysis **for** object **oriented languages. Application to Java**
Author: Blanchet, Bruno
Corporate Source: INRIA Rocquencourt, Le Chesnay, Fr
Conference Title: Proceedings of the 1999 Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA-99
Conference Location: Denver, CO, USA Conference Date: 20991101-20991105
E.I. Conference No.: 56092
Source: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA v 34 n 10 Oct 1999. ACM, New York, NY, USA. p 20-34
Publication Year: 1999
CODEN: 002376
Language: English
Document Type: CA; (Conference Article) Treatment: A; (Applications); T ; (Theoretical)
Journal Announcement: 0104W3
Abstract: **Escape analysis** is a static analysis that determines whether the lifetime of data exceeds its static scope. The main originality of our **escape analysis** is that it determines precisely the effect of assignments, which is necessary to apply it to **object** oriented languages with promising results, whereas previous work applied it to functional languages and were very imprecise on assignments. Our implementation analyses the full Java Language. We have applied our analysis to **stack**

allocation and synchronization elimination. We manage to **stack** allocate 13% to 95% of data, eliminate more than 20% of synchronizations on most programs (94% and 99% on two examples) and get up to 44% speedup (21% on average). Our detailed experimental study on large programs shows that the improvement comes more from the decrease of the **garbage collection** and allocation times than from improvements on data locality, contrary to what happened for ML. (Author abstract) 31 Refs.

Descriptors: Data reduction; Java programming language; **Object** oriented programming; **Storage** allocation (computer); Algorithms; Computational complexity; C (programming language); Program compilers

Identifiers: **Escape analysis** ; Functional languages; **Stack** allocation; Synchronization elimination

Classification Codes:

723.1.1 (Computer Programming Languages)

723.2 (Data Processing); 723.1 (Computer Programming); 722.1 (Data Storage, Equipment & Techniques); 723.5 (Computer Applications); 721.1 (Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory)

723 (Computer Software); 722 (Computer Hardware); 721 (Computer Circuits & Logic Elements)

72 (COMPUTERS & DATA PROCESSING)

31/5/5 (Item 5 from file: 8)

05804674 E.I. No: EIP00025069923

Title: Escape analysis **for Java**

Author: Choi, Jong-Deok; Gupta, Manish; Serrano, Mauricio; Sreedhar, Vugranam C.; Midkiff, Sam

Corporate Source: IBM T.J. Watson Research Cent, Yorktown Heights, NY, USA

Conference Title: Proceedings of the 1999 Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA-99

Conference Location: Denver, CO, USA Conference Date: 20991101-20991105

E.I. Conference No.: 56092

Source: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA v 34 n 10 Oct 1999. ACM, New York, NY, USA. p 1-19

Publication Year: 1999

CODEN: 002376

Language: English

Document Type: CA; (Conference Article) Treatment: T; (Theoretical); X; (Experimental)

Journal Announcement: 0104W3

Abstract: This paper presents a simple and efficient data flow algorithm for **escape analysis** of **objects** in Java programs to determine (i) if an **object** can be allocated on the **stack** ; (ii) if an **object** is accessed only by a single thread during its lifetime, so that synchronization operations on that **object** can be removed. We introduce a new program abstraction for **escape analysis** , the connection graph, that is used to establish reachability relationships between **objects** and **object** references. We show that the connection graph can be summarized for each method such that the same summary **information** may be used effectively in different calling contexts. We present an interprocedural algorithm that uses the above property to efficiently compute the connection graph and identify the non-escaping **objects** for methods and threads. The experimental results, from a prototype implementation of our framework in the IBM High Performance Compiler for Java, are very promising. The percentage of **objects** that may be allocated on the **stack** exceeds 70% of all dynamically created **objects** in three out of the ten benchmarks (with a median of 19%), 11% to 92% of all lock operations are eliminated in those ten programs (with a median of 51%), and the overall execution time reduction ranges from 2% to 23% (with a median of 7%) on a 333 MHz PowerPC workstation with 128 MB **memory** . (Author abstract) 25 Refs.

Descriptors: Data flow analysis; Java programming language; Algorithms; Program compilers; **Storage** allocation (computer); Procedure oriented languages; Computer workstations
Identifiers: **Escape analysis** ; Program abstraction; Connection graph; Interprocedural algorithms
Classification Codes:
723.1.1 (Computer Programming Languages)
723.1 (Computer Programming); 723.5 (Computer Applications); 722.1 (Data Storage, Equipment & Techniques); 722.4 (Digital Computers & Systems)
723 (Computer Software); 722 (Computer Hardware)
72 (COMPUTERS & DATA PROCESSING)

31/5/6 (Item 6 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05382016 E.I. No: EIP99094777340
Title: Efficient implementation of Java's Remote Method Invocation
Author: Maassen, Jason; van Nieuwpoort, Rob; Veldema, Ronald; Bal, Henri E.; Plaat, Aske
Corporate Source: Vrije Universiteit, Amsterdam, Neth
Conference Title: Proceedings of the 1999 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) aspart of the Federated Computing Research Conference(FCRC'99)
Conference Location: Atlanta, GA, USA Conference Date: 19990504-19990506
Sponsor: ACM SIGPLAN
E.I. Conference No.: 55458
Source: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 1999. p 173-182
Publication Year: 1999
CODEN: 002167
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 9911W3
Abstract: Java offers interesting opportunities for parallel computing. In particular, Java Remote Method Invocation provides an unusually flexible kind of Remote Procedure Call. Unlike RPC, RMI supports polymorphism, which requires the system to be able to download remote classes into a running application. Sun's RMI implementation achieves this kind of flexibility by passing around **object** type **information** and processing it at run time, which causes a major run time overhead. Using Sun's JDK 1.1.4 on a Pentium Pro/Myrinet cluster, for example, the latency for a null RMI (without parameters or a **return** value) is 1228 mu sec, which is about a factor of 40 higher than that of a user-level RPC. In this paper, we study an alternative approach for implementing RML based on native compilation. This approach allows for better optimization, eliminate the need for processing of type **information** at run time, and makes a light weight communication protocol possible. We have built a Java system based on a native compiler, which supports both compile time and **run time generation** of marshallers. We find that almost all of the run time overhead of RMI can be pushed to compile time. With this approach, the latency of a null RMI is reduced to 34 mu sec, while still supporting polymorphic RMIs (and allowing interoperability with other JVMs). (Author abstract) 33 Refs.
Descriptors: *Parallel processing systems; Java programming language; Subroutines; Data structures; Program compilers; Data communication systems ; Network protocols; Computer systems programming; Response time (computer systems)
Identifiers: Remote method invocation (RMI); Remote procedure call (RPC)
Classification Codes:
723.1.1 (Computer Programming Languages)
722.4 (Digital Computers & Systems); 723.1 (Computer Programming); 723.2 (Data Processing)
722 (Computer Hardware); 723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

31/5/7 (Item 7 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05289984 E.I. No: EIP99054678652
Title: Pure Java-based streaming MPEG player
Author: Tolba, Osama; Briceno, Hector; McMillan, Leonard
Corporate Source: MIT, Cambridge, MA, USA
Conference Title: Proceedings of the 1998 Multimedia Systems and Applications
Conference Location: Boston, MA, USA Conference Date: 19981102-19981104
Sponsor: SPIE
E.I. Conference No.: 55042
Source: Proceedings of SPIE - The International Society for Optical Engineering v 3528 1999. p 216-224
Publication Year: 1999
CODEN: PSISDG ISSN: 0277-786X
Language: English
Document Type: JA; (Journal Article) Treatment: G; (General Review); T; (Theoretical)
Journal Announcement: 9907W3
Abstract: We present a pure Java-based streaming MPEG-1 video player. By implementing the player entirely in Java, we guarantee its functionality across platforms within any Java-enabled web browsers, without the need for native libraries. This allows greater use of MPEG video sequences, because the users will no longer need to pre-install any software to display video, beyond Java compatibility. This player features a novel forward-mapping IDCT algorithm that allows it to play locally stored, CIF-sized (352 multiplied by 288) video sequences at 11 frames per second, when run on a personal computer with Java ` **Just - in - Time** ' **compiler** . The IDCT algorithm can run with greater speed when the sequence is viewed at reduced size; e.g., performing approximately one quarter the amount of computation when the user resizes the sequence to one half its original width and height. We are able to play video streams stored anywhere on the Internet with acceptable performance using a proxy server, eliminating the need for large-capacity auxiliary storage. Thus, the player is well suited to small devices, such as digital TV set-top decoders, requiring little more memory than is required for three video frames. Because of our modular design, it is possible to assemble multiple video streams from remote sources and present them simultaneously to the viewers (i.e. picture-in-a-picture style), subject to network and local performance limitations. The same modular system can further provide viewers with their own customized view of each session; e.g., **moving** and resizing the video display window dynamically, and selecting their preferred set of video controls. (Author abstract) 12 Refs.
Descriptors: *Multimedia systems; Java programming language; Web browsers ; Algorithms; Program compilers; Internet; Digital television; Standards
Identifiers: Motion picture experts group (MPEG) standards; Video players
Classification Codes:
723.1.1 (Computer Programming Languages)
723.5 (Computer Applications); 723.1 (Computer Programming); 716.4 (Television Systems & Equipment); 902.2 (Codes & Standards)
723 (Computer Software); 921 (Applied Mathematics); 716 (Radar, Radio & TV Electronic Equipment); 902 (Engineering Graphics & Standards)
72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS); 71 (ELECTRONICS & COMMUNICATIONS); 90 (GENERAL ENGINEERING)

31/5/8 (Item 8 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05050322 E.I. No: EIP98064269756
Title: Escape analysis : Correctness proof, implementation and experimental results

Author: Blanchet, Bruno
Corporate Source: INRIA Rocquencourt, Le Chesnay, Fr
Conference Title: Proceedings of the 1998 25th ACM SIGPLAN SIGACT Symposium on Principles of Programming Languages
Conference Location: San Diego, CA, USA Conference Date: 19980119-19980121
Sponsor: ACM
E.I. Conference No.: 48540
Source: Conference Record of the Annual ACM Symposium on Principles of Programming Languages 1998. ACM, New York, NY, USA. p 25-37
Publication Year: 1998
CODEN: CRLADV ISSN: 0730-8566
Language: English
Document Type: CA; (Conference Article) Treatment: T; (Theoretical); X; (Experimental)
Journal Announcement: 9808W4
Abstract: We describe an **escape analysis** , used to determine whether the lifetime of data exceeds its static scope. We give a new correctness proof starting directly from a semantics. Contrary to previous proofs, it takes into account all the features of functional languages, including imperative features and polymorphism. The analysis has been designed so that it can be implemented under the small complexity bound of O(nlog**2 n) where n is the size of the analyzed program. We have included it in the Caml Special Light compiler (an implementation of ML), and applied it to very large programs. We plan to apply these techniques to the Java programming language. **Escape analysis** has been applied to **stack** allocation. We improve the optimization technique by determining minimal lifetime for **stack** allocated data, and using inlining. We manage to **stack** allocate 25% of data in the theorem prover Coq. We analyzed the effect of this optimization, and noticed that its main effect is to improve data locality, which is important for efficiency. (Author abstract) 38 Refs.
Descriptors: Formal languages; Computational linguistics; Computational complexity; Program compilers; High level languages; **Storage** allocation (computer); Optimization; Theorem proving
Identifiers: **Escape analysis** ; Polymorphism
Classification Codes:
723.1.1 (Computer Programming Languages)
721.1 (Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory); 723.1 (Computer Programming); 722.1 (Data Storage, Equipment & Techniques); 921.5 (Optimization Techniques)
721 (Computer Circuits & Logic Elements); 723 (Computer Software); 722 (Computer Hardware); 921 (Applied Mathematics)
72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

31/5/9 (Item 9 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

03622450 E.I. No: EIP93030723481
Title: Escape analysis **on lists**
Author: Park, Young Gil; Goldberg, Benjamin
Corporate Source: New York Univ, New York, NY, USA
Conference Title: Proceedings of the ACM SIGPLAN '92 Conference on Programming Language Design and Implementation
Conference Location: San Francisco, CA, USA Conference Date: 19920617
Sponsor: ACM
E.I. Conference No.: 17942
Source: SIGPLAN Notices (ACM Special Interest Group on Programming Languages) 1992. Publ by ACM, New York, NY, USA. p 116-127
Publication Year: 1992
CODEN: SINODQ ISSN: 0362-1340 ISBN: 0-89791-475-9
Language: English
Document Type: CA; (Conference Article) Treatment: T; (Theoretical)
Journal Announcement: 9306W3
Abstract: Higher order functional programs constantly allocate **objects**

dynamically. These **objects** are typically cons cells, closures, and records and are generally allocated in the **heap** and reclaimed later by some **garbage collection** process. This paper describes a compile time analysis, called **escape analysis** , for determining the lifetime of dynamically created **objects** in higher order functional programs, and describes optimizations that can be performed, based on the analysis, to improve **storage** allocation and reclamation of such **objects** . In particular, our analysis can be applied to programs manipulating lists, in which case optimizations can be performed to allow cons cells in spines of lists to be either reclaimed immediately or reused without incurring any **garbage collection** overhead. In a previous paper on **escape analysis** , we had left open the problem of performing **escape analysis** on lists. **Escape analysis** simply determines when the argument (or some part of the argument) to a function call is **returned** by that call. This simple piece of **information** turns out to be sufficiently powerful to allow **stack** allocation of **objects** , compile-time **garbage collection** , reduction of run-time **storage** reclamation overhead, and other optimizations that are possible when the lifetimes of **objects** can be computed statically. Our approach is to define a high-level non-standard semantics that, in many ways, is similar to the standard semantics and captures the escape behavior caused by the constructs in a functional language. The advantage of our analysis lies in its conceptual simplicity and portability (i.e. no assumption is made about an underlying abstract machine). (Author abstract) 19 Refs.

Descriptors: *Computer programming; Program compilers

Identifiers: **Escape analysis** ; Compile time analysis; Higher order functional programs

Classification Codes:
723.1 (Computer Programming)
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

31/5/11 (Item 2 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online

01234090 ORDER NO: AAD92-22908
SEMANTIC ANALYSES FOR STORAGE **MANAGEMENT OPTIMIZATIONS IN FUNCTIONAL LANGUAGE IMPLEMENTATIONS**

Author: PARK, YOUNG-GIL
Degree: PH.D.
Year: 1992
Corporate Source/Institution: NEW YORK UNIVERSITY (0146)
Adviser: BENJAMIN GOLDBERG
Source: VOLUME 53/03-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 1471. 248 PAGES
Descriptors: COMPUTER SCIENCE
Descriptor Codes: 0984

One of the major overheads in implementing functional languages is the **storage** management overhead due to dynamic allocation and automatic reclamation of indefinite-extent **storage** . This dissertation investigates the problems of statically inferring lifetime **information** about dynamically-allocated **objects** in higher-order polymorphic functional languages, both strict and non-strict, and of applying that **information** to reduce the **storage** management overhead.

We have developed a set of compile-time semantic analyses for a higher-order, monomorphic, strict functional language based on denotational semantics and abstract interpretation. They are (1) **escape analysis** , which provides **information** about the relative lifetimes of **objects** such as arguments and local **objects** defined within a function with respect to an activation of the function call, (2) refined **escape analysis** which, as a refinement of **escape analysis** , provides **information** about the lifetime of components of aggregate structures, and (3) reference **escape analysis** which provides **information** about the relative lifetimes of references created within a function with respect to an activation of the

function.

We also have developed a compile-time semantic analysis called order-of-demand analysis for higher-order, monomorphic, non-strict functional languages, which provides **information** about the order in which the values of bound variables are demanded and thus allows one to compute a range of **information** including strictness, evaluation-order, and evaluation-status **information** .

Using the notion of polymorphic invariance, we describe a method for analyzing a polymorphic language by using the analyses for a monomorphic language. We then extend those analyses for a strict language to a non-strict language using non-strict program transformation and evaluation-status **information** .

Based on statically inferred escape **information** , we propose a combination of **storage** management optimization techniques including **stack** allocation, explicit reclamation, inplace reuse, reference counting elimination, block allocation/reclamation, and improving generational **garbage collection** .

31/5/12 (Item 1 from file: 65)
DIALOG(R)File 65:Inside Conferences

03221327 INSIDE CONFERENCE ITEM ID: CN034070817
Fast Escape Analysis **and** Stack **Allocation for** Object **-Based Programs**
Gay, D.; Steensgaard, B.
LECTURE NOTES IN COMPUTER SCIENCE, 2000; (NO) 1781 P: 82-93
Berlin, Springer, 2000
ISSN: 0302-9743 ISBN: 354067263X
LANGUAGE: English DOCUMENT TYPE: Conference Papers
CONFERENCE EDITOR(S): Watt, D. A.

BRITISH LIBRARY ITEM LOCATION: 5180.185000
DESCRIPTORS: compiler construction; ETAPS; CC

31/5/13 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

6900913 INSPEC Abstract Number: C2001-05-4240-010
Title: Programmable environment calculus as theory of dynamic software evolution
Author(s): Nishizaki, S.
Author Affiliation: Dept. of Comput. Sci., Tokyo Inst. of Technol., Japan
Conference Title: Proceedings International Symposium on Principles of Software Evolution p.221-5
Editor(s): Katayama, T.; Tamai, T.; Yonezaki, N.
Publisher: IEEE Comput. Soc, Los Alamits, CA, USA
Publication Date: 2000 Country of Publication: USA ix+321 pp.
ISBN: 0 7695 0906 1 Material Identity Number: XX-2001-00522
U.S. Copyright Clearance Center Code: 0 7695 0906 1/2000/$10.00
Conference Title: Proceedings International Symposium on Principles of Software Evolution
Conference Sponsor: Japan Adv. Inst. Sci. & Technol. (JAIST); Grand-In-Aid for Priority Areas `Principles for Evolutionary Software' (Minstr. Educ., Japan); Res. for the Future Program `Software Dev. Methodology' (JSPS-RFTF)
Conference Date: 1-2 Nov. 2000 Conference Location: Kanazawa, Japan
Language: English Document Type: Conference Paper (PA)
Treatment: Theoretical (T)
Abstract: Software evolution is one of the most important features in advanced computer systems, and the significance of its theoretical study is acknowledged. Software evolution is divided into two categories: static and dynamic evolution. Static evolution involves changes in a software system that occur before compilation; a typical example is a software update. On

the other hand, dynamic evolution involves changes in the execution time of a software system. The dynamic library mechanism in operating systems and **dynamic class loading** in Java are types of dynamic evolution. An environment represents a mapping of variables onto values. We have studied the lambda calculus with first-class environments (called the environment calculus). With this, we can treat environments as first-class citizens: environment values can be passed as parameters and **returned** as resultant values. The first-class environments are formalized according to the idea of explicit substitutions. This paper proposes programmable environments, as a further extension of first-class environments, which provide a computational mechanism allowing first-class environments to be treated as functions mapping variables onto their bound values. Conversely, such functions can also be treated as first-class environments. Programmable environments allow us to operate meta-level name spaces directly, and they enable us to model the dynamic evolution mechanism. (17 Refs)

Subfile: C

Descriptors: lambda calculus; programming theory; software maintenance

Identifiers: programmable environment calculus; dynamic software evolution; static evolution; execution time changes; dynamic library mechanism; operating systems; **dynamic class loading** ; Java; variables mapping; lambda calculus; first-class environments; environment values; parameter passing; explicit substitutions; computational mechanism; bound values; meta-level name spaces

Class Codes: C4240 (Programming and algorithm theory); C4210 (Formal logic); C6110B (Software engineering techniques)

31/5/14 (Item 2 from file: 2)

DIALOG(R)File 2:INSPEC

6793465 INSPEC Abstract Number: C2001-02-6150C-004

Title: Software profiling for hot path prediction: less is more

Author(s): Duesterwald, E.; Bala, V.

Author Affiliation: Hewlett-Packard Labs., Cambridge, MA, USA

Journal: Operating Systems Review Conference Title: Oper. Syst. Rev. (USA) vol.34, no.5 p.202-11

Publisher: ACM,

Publication Date: Dec. 2000 Country of Publication: USA

CODEN: OSRED8 ISSN: 0163-5980

SICI: 0163-5980(200012)34:5L.202:SPPP;1-8

Material Identity Number: O043-2000-006

Conference Title: ASPLOS-IX. Ninth International Conference on Architectural Support for Programming Languages and Operating Systems

Conference Sponsor: ACM

Conference Date: 12-15 Nov. 2000 Conference Location: Cambridge, MA, USA

Language: English Document Type: Conference Paper (PA); Journal Paper (JP)

Treatment: Practical (P)

Abstract: Recently, there has been a growing interest in exploiting profile **information** in adaptive systems such as **just - in - time compilers** , dynamic optimizers and, binary translators. In this paper, we show that sophisticated software profiling schemes that provide highly accurate **information** in an offline setting are ill-suited for these **dynamic code generation** systems. We experimentally demonstrate that hot path predictions must be made early in order to control. The rising cost of missed opportunity that result from the prediction delay. We also show that existing sophisticated path profiling schemes, if used in an online setting, offer no prediction advantages over simpler schemes that exhibit much lower runtime overheads. Based on these observation we developed a new low-overhead software profiling scheme for hot path prediction. Using an abstract metric we compare our scheme to path profile based prediction and show that our scheme achieves comparable prediction quality. In our second set of experiments we include runtime overhead and evaluate the performance of our scheme in a realistic application: Dynamo,

a dynamic optimization system. The results show that our prediction scheme clearly outperforms path profile based prediction and thus confirm that less profiling as exhibited in our scheme will actually lead to **move** effective hot path prediction. (21 Refs)
Subfile: C
Descriptors: performance evaluation; program compilers
Identifiers: hot path prediction; profile **information** ; adaptive systems ; **just - in - time compilers** ; software profiling; code generation; low-overhead software profiling
Class Codes: C6150C (Compilers, interpreters and other processors)

31/5/15 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC

6726125 INSPEC Abstract Number: C2000-11-6120-011
Title: Annotating Java class files with virtual registers for performance
Author(s): Jones, J.; Kamin, S.
Author Affiliation: Dept. of Comput. Sci., Illinois Univ., Urbana, IL, USA
Journal: Concurrency: Practice and Experience Conference Title: Concurrency, Pract. Exp. (UK) vol.12, no.6 p.389-406
Publisher: Wiley,
Publication Date: May 2000 Country of Publication: UK
CODEN: CPEXEI ISSN: 1040-3108
SICI: 1040-3108(200005)12:6L.389:AJCF;1-G
Material Identity Number: N824-2000-006
U.S. Copyright Clearance Center Code: 1040-3108/2000/$30.00
Conference Title: Proceedings of the 1999 Association for Computing Machinery Conference on Java Grande
Conference Sponsor: ACM
Conference Date: 12-14 June 1999 Conference Location: San Francisco, CA, USA
Language: English Document Type: Conference Paper (PA); Journal Paper (JP)
Treatment: Practical (P)
Abstract: The Java .class file is a compact encoding of programs for a **stack** based virtual machine. It is intended for use in a networked environment, which requires machine independence and minimized consumption of network bandwidth. However, as in all interpreted virtual machines, performance does not match that of code generated for the target machine. We propose verifiable, machine-independent annotations to the Java class file to bring the quality of the code generated by a " **just - in - time** " **compiler** closer to that of an optimizing compiler without a significant increase in code generation time. This division of labor has expensive machine-independent analysis performed offline and inexpensive machine-dependent code generation performed on the client. We call this phenomenon "super-linear analysis and linear exploitation". These annotations were designed mindful of the concurrency features of the Java language. The authors report results from their machine-independent, prioritized register assignment. They also discuss other possible annotations. (19 Refs)
Subfile: C
Descriptors: Java; program compilers; **storage** allocation; virtual machines; virtual **storage**
Identifiers: Java class file annotation; virtual registers; compact program encoding; **stack** based virtual machine; networked environment; machine independence; minimized consumption; network bandwidth; interpreted virtual machines; code generation; verifiable machine-independent annotations; **just - in - time compiler** ; optimizing compiler; machine-independent analysis; machine-dependent code generation; super-linear analysis; linear exploitation; concurrency features; Java language; prioritized register assignment
Class Codes: C6120 (File organisation); C6110J (Object-oriented programming); C6140D (High level languages); C6150N (Distributed systems

software); C7430 (Computer engineering); C6150C (Compilers, interpreters and other processors)

31/5/16 (Item 4 from file: 2)
DIALOG(R)File 2:INSPEC

6428949 INSPEC Abstract Number: C2000-01-6120-013
Title: Escape analysis **for** object **oriented languages. Application to Java/sup TM/**
Author(s): Blanchet, B.
Author Affiliation: Inst. Nat. de Recherche en Inf. et Autom., Le Chesnay, France
Journal: SIGPLAN Notices Conference Title: SIGPLAN Not. (USA) vol.34, no.10 p.20-34
Publisher: ACM,
Publication Date: Oct. 1999 Country of Publication: USA
CODEN: SINODQ ISSN: 0362-1340
SICI: 0362-1340(199910)34:10L.20:EAOO;1-O
Material Identity Number: S202-1999-011
Conference Title: Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99)
Conference Sponsor: ACM
Conference Date: 1-5 Nov. 1999 Conference Location: Denver, CO, USA
Language: English Document Type: Conference Paper (PA); Journal Paper (JP)
Treatment: Practical (P)
Abstract: **Escape analysis** is a static analysis that determines whether the lifetime of data exceeds its static scope. The main originality of our **escape analysis** is that it determines precisely the effect of assignments, which is necessary to apply it to **object** oriented languages with promising results, whereas previous work applied it to functional languages and were very imprecise on assignments. Our implementation analyses the full Java/sup TM/ Language. We have applied our analysis to **stack** allocation and synchronization elimination. We manage to **stack** allocate 13% to 95% of data, eliminate more than 20% of synchronizations on most programs (94% and 99% on two examples) and get up to 44% speedup (21% on average). Our detailed experimental study on large programs shows that the improvement comes more from the decrease of the **garbage collection** and allocation times than from improvements on data locality, contrary to what happened for ML. (31 Refs)
Subfile: C
Descriptors: Java; **object** -oriented languages; **storage** allocation; system monitoring
Identifiers: **escape analysis** ; static analysis; data lifetime; **object** oriented languages; Java; **stack** allocation; synchronization elimination; **garbage collection** ; data locality
Class Codes: C6120 (File organisation); C6140D (High level languages); C6150G (Diagnostic, testing, debugging and evaluating systems)

31/5/17 (Item 5 from file: 2)
DIALOG(R)File 2:INSPEC

6428948 INSPEC Abstract Number: C2000-01-6110J-026
Title: Escape analysis **for Java**
Author(s): Jong-Deok Choi; Gupta, M.; Serrano, M.; Sreedhar, V.C.; Midkiff, S.
Author Affiliation: IBM Thomas J. Watson Res. Center, Yorktown Heights, NY, USA
Journal: SIGPLAN Notices Conference Title: SIGPLAN Not. (USA) vol.34, no.10 p.1-19
Publisher: ACM,

Publication Date: Oct. 1999 Country of Publication: USA
CODEN: SINODQ ISSN: 0362-1340
SICI: 0362-1340(199910)34:10L.1:EAJ;1-1
Material Identity Number: S202-1999-011
Conference Title: Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99)
Conference Sponsor: ACM
Conference Date: 1-5 Nov. 1999 Conference Location: Denver, CO, USA
Language: English Document Type: Conference Paper (PA); Journal Paper (JP)
Treatment: Practical (P)
Abstract: This paper presents a simple and efficient data flow algorithm for **escape analysis** of **objects** in Java programs to determine (i) if an **object** can be allocated on the **stack** ; (ii) if an **object** is accessed only by a single thread during its lifetime, so that synchronization operations on that **object** can be removed. We introduce a new program abstraction for **escape analysis** , the connection graph, that is used to establish reachability relationships between **objects** and **object** references. We show that the connection graph can be summarized for each method such that the same summary **information** may be used effectively in different calling contexts. We present an interprocedural algorithm that uses the above property to efficiently compute the connection graph and identify the non-escaping **objects** for methods and threads. The experimental results, from a prototype implementation of our framework in the IBM High Performance Compiler for Java, are very promising. The percentage of **objects** that may be allocated on the **stack** exceeds 70% of all dynamically created **objects** in three out of the ten benchmarks (with a median of 19%), 11% to 92% of all lock operations are eliminated in those ten programs (with a median of 51%), and the overall execution time reduction ranges from 2% to 23% (with a median of 7%) on a 333 MHz PowerPC workstation with 128 MB **memory** . (25 Refs)
Subfile: C
Descriptors: data flow analysis; Java; **object** -oriented programming; program compilers; reachability analysis; **storage** allocation
Identifiers: data flow algorithm; **object escape analysis** ; Java programs; **object** allocation; **stack** ; synchronization operations; program abstraction; connection graph; reachability relationships; **object** references; summary **information** ; calling contexts; interprocedural algorithm; nonescaping **objects** ; threads; IBM High Performance Compiler; dynamically created **objects** ; benchmarks; lock operations; PowerPC workstation; 333 MHz
Class Codes: C6110J (Object-oriented programming); C6150G (Diagnostic, testing, debugging and evaluating systems); C6120 (File organisation); C6150C (Compilers, interpreters and other processors)
Numerical Indexing: frequency 3.33E+08 Hz

31/5/18 (Item 6 from file: 2)
DIALOG(R)File 2:INSPEC

6412185 INSPEC Abstract Number: C2000-01-5320C-004
Title: Assessing the value of virtual tape technology: what it can do for you
Journal: Storage Management vol.7, no.1 p.3-14
Publisher: Demand Technol,
Publication Date: 1999 Country of Publication: USA
CODEN: MFSMER ISSN: 1090-0799
SICI: 1090-0799(1999)7:1L.3:AVVT;1-Q
Material Identity Number: H142-1999-003
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: Virtual tape hardware from **IBM** , SUTMYN, and StorageTek is necessary to utilize current generation high density tape drives effectively in an automated tape environment. The original products were **earmarked** to solving the small files on tape problem for batch

processing. Now we need bulked-up virtual tape devices on steroids to **stack** MVS logical disk-to-tape backup volumes on 50 GB capacity tape cartridges. The paper considers whether any of the products on the market today meet these requirements. (0 Refs)
Subfile: C
Descriptors: batch processing (computers); magnetic tape **storage**
Identifiers: virtual tape technology; **IBM** ; SUTMYN; StorageTek; high density tape drives; automated tape environment; batch processing; MVS; logical disk-to-tape backup
Class Codes: C5320C (Storage on moving magnetic media)

31/5/19 (Item 7 from file: 2)
DIALOG(R)File 2:INSPEC

6410352 INSPEC Abstract Number: C2000-01-6115-002
Title: XML and CORBA
Author(s): Hamstra, D.
Journal: Dr. Dobb's Journal vol.24, no.11 p.98-100
Publisher: Miller Freeman,
Publication Date: Nov. 1999 Country of Publication: USA
CODEN: DDJSDM ISSN: 1044-789X
SICI: 1044-789X(199911)24:11L.98:C;1-C
Material Identity Number: B719-1999-010
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P); Product Review (R)
Abstract: The XML/IT toolkit from CareFlow allows you to automatically tag the results **returned** from calls to CORBA-based services, and then format them using XML. It also includes utilities that support the conversion of XML-tagged documents to Java structures, and vice versa. In addition to assuming the existence of CORBA-based back-end services, XML/IT assumes the use of interface definition language (IDL) to **Java compilers** to generate Java stubs and skeletons, and uses CORBA's Dynamic Invocation Interface (DII) standards specification. The main client program routine (DIICall) can be embedded inside other Java-based clients, CGI scripts or Java servlets. (0 Refs)
Subfile: C
Descriptors: distributed **object** management; hypermedia markup languages ; Java; software reviews; software tools; utility programs
Identifiers: XML/IT toolkit; CareFlow; automatic tagging; CORBA-based services; results formatting; utilities; XML-tagged document conversion; Java structures; back-end services; interface definition language; IDL to **Java compilers** ; Java stubs; Java skeletons; Dynamic Invocation Interface ; standards specification; embedded client program routine; DIICall; Java-based clients; CGI scripts; Java servlets
Class Codes: C6115 (Programming support); C6130D (Document processing techniques); C6130M (Multimedia); C6140D (High level languages); C6110J (Object-oriented programming); C6150N (Distributed systems software); C6150E (General utility programs)

31/5/20 (Item 8 from file: 2)
DIALOG(R)File 2:INSPEC

6409837 INSPEC Abstract Number: C1999-12-6150C-051
Title: LaTTe: a Java VM just - in - time compiler with fast and efficient register allocation
Author(s): Byung-Sun Yang; Soo-Mook Moon; Seongbae Park; Junpyo Lee; SeungIl Lee; Jinpyo Park; Chung, Y.C.; Suhyun Kim; Ebcioglu, K.; Altman, E.
Author Affiliation: Seoul Nat. Univ., South Korea
Conference Title: 1999 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00425) p.128-38
Publisher: IEEE Comput. Soc, Los Alamitos, CA, USA

Publication Date: 1999 Country of Publication: USA xv+321 pp.
ISBN: 0 7695 0425 6 Material Identity Number: XX-1999-02922
U.S. Copyright Clearance Center Code: 0 7695 0425 6/99/$10.00
Conference Title: 1999 International Conference on Parallel Architectures and Compilation Techniques
Conference Sponsor: IFIP; IEEE Comput. Soc
Conference Date: 12-16 Oct. 1999 Conference Location: Newport Beach, CA, USA
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: For network computing on desktop machines, fast execution of Java bytecode programs is essential because these machines are expected to run substantial application programs written in Java. Higher Java performance can be achieved by just-in-time (**JIT**) **compilers** which translate the **stack** -based bytecode into register-based machine code on demand. One crucial problem in Java JIT compilation is how to map and allocate **stack** entries and local variables into registers efficiently and quickly, so as to improve the Java performance. This paper introduces LaTTe, a Java **JIT compiler** that performs fast and efficient register mapping and allocation for RISC machines. LaTTe first translates the bytecode into pseudo RISC code with symbolic registers, which is then register allocated while coalescing those copies corresponding to pushes and pops between local variables and the **stack** . The LaTTe JVM also includes an enhanced object model, a lightweight monitor, a fast mark-and-sweep garbage collector, and an on-demand exception handling mechanism, all of which are closely coordinated with LaTTe's JIT compilation. (12 Refs)
Subfile: C
Descriptors: exception handling; Java; network computers; object-oriented programming; program compilers; reduced instruction set computing; software performance evaluation; **storage** allocation
Identifiers: LaTTe; Java; **just - in - time compiler** ; register allocation; network computing; desktop machines; **stack** -based bytecode; register-based machine code; performance; register mapping; RISC; object model; lightweight monitor; mark-and-sweep garbage collector; exception handling
Class Codes: C6150C (Compilers, interpreters and other processors); C6110J (Object-oriented programming); C6140D (High level languages); C6120 (File organisation)

31/5/21 (Item 9 from file: 2)
DIALOG(R)File 2:INSPEC

6390819 INSPEC Abstract Number: C1999-12-6140D-010
Title: Is Java suitable for portable high-performance computing? Preliminary reports on benchmarking different Java platforms
Author(s): Matsuoka, S.; Itou, S.
Author Affiliation: Dept. of Math. & Comput. Sci., Tokyo Inst. of Technol., Japan
Conference Title: Object-Oriented Technology. ECOOP'98 Workshop Reader. ECOOP'98 Workshops, Demos, and Posters. Proceedings p.460-1
Editor(s): Demeyer, S.; Bosch, J.
Publisher: Springer-Verlag, Berlin, Germany
Publication Date: 1998 Country of Publication: Germany xxii+573 pp.
ISBN: 3 540 65460 7 Material Identity Number: XX-1999-01939
Conference Title: Object-Oriented Technology. ECOOP'98 Workshop Reader
Conference Date: 20-24 July 1998 Conference Location: Brussels, Belgium
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P); Experimental (X)
Abstract: We are working on making Java a suitable portable platform for high-performance computing. This paper reports on some initial results in our benchmarking and performance analysis results. We found that traditional user-level optimizations for numerical computing, such as strip

mining, do not always properly apply, due to the unpredictability of both the **JIT compiler** and the VM (virtual machine) run-time system. Most traditional compilers for high-performance computing on RISC-based processors apply various optimization techniques to allow efficient scheduling of in-processor resources for fine-grained parallelism, and also to utilize the **cache** as much as possible to overcome the **memory** wall problem. However, it is not obvious whether such optimization techniques, especially those performed at the source-code level by the compiler or a user, will be effective for Java language implementations. Some of the potential problems could be categorized into language specification problems, compiler problems and run-time problems: aliasing of multi-dimensional arrays; array bounds checking; intervention of **stack** -based byte code; compilers not tailored for high-performance computing; exception handling; the restricted space/time resources of **JIT compilers** ; unpredictable **memory** allocation; multithreading overhead; low performance of remote method invocation (RMI); VM-based execution restrictions; and other reasons. Given these observations, it could be said that, despite Java's good portability in the sense that it works, it could be difficult to achieve portability in performance. (2 Refs)

Subfile: C

Descriptors: **cache storage** ; Java; optimising compilers; reduced instruction set computing; software performance evaluation; software portability; virtual machines

Identifiers: portable high-performance computing; Java platform benchmarking; performance analysis; user-level optimizations; numerical computing; strip mining; virtual machine run-time system; RISC-based processors; in-processor resource scheduling; fine-grained parallelism; **cache** utilization; **memory** wall problem; source-code level optimization; language specification problems; compiler problems; run-time problems; multi-dimensional array aliasing; array bounds checking; **stack** -based byte code; exception handling; restricted space/time resources; **JIT compilers** ; unpredictable **memory** allocation; multithreading overhead; remote method invocation; execution restrictions; software portability

Class Codes: C6140D (High level languages); C6110J (Object-oriented programming); C6150N (Distributed systems software); C6110B (Software engineering techniques); C6150C (Compilers, interpreters and other processors); C5470 (Performance evaluation and testing)

31/5/22 (Item 10 from file: 2)

DIALOG(R)File 2:INSPEC

6316542 INSPEC Abstract Number: C1999-09-6150C-006

Title: Transient variable caching in Java's stack **-based intermediate representation**

Author(s): Tyma, P.

Author Affiliation: Syracuse Univ., Euclid, OH, USA

Journal: Scientific Programming vol.7, no.2 p.157-66

Publisher: IOS Press,

Publication Date: 1999 Country of Publication: Netherlands

CODEN: SCIPEV ISSN: 1058-9244

SICI: 1058-9244(1999)7:2L.157:TVCJ;1-8

Material Identity Number: H288-1999-005

U.S. Copyright Clearance Center Code: 1058-9244/99/$8.00

Language: English Document Type: Journal Paper (JP)

Treatment: Practical (P)

Abstract: Java's **stack** based intermediate representation (IR) is typically coerced to execute on register based architectures. Unoptimized compiled code dutifully replicates transient variable usage designated by the programmer and common optimization practices tend to introduce further usage (i.e., CSE, Loop-invariant Code Motion, etc.). On register based machines, often transient variables are **cached** within registers (when available) saving the expense of actually accessing **memory** . Unfortunately, in **stack** based environments because of the need to push and pop the transient values, further performance improvement is possible.

The paper presents Transient Variable Caching (TVC), a technique for eliminating transient variable overhead whenever possible. This optimization would find a likely home in optimizers attached to the back of popular **Java compilers** . Side effects of the algorithm include significant instruction reordering and introduction of many **stack** manipulation operations. This combination has proven to greatly impede the ability to decompile **stack** based IR code sequences. The code that results from the transform is faster, smaller, and greatly impedes decompilation. (26 Refs)

Subfile: C

Descriptors: abstract data types; **cache storage** ; Java; program compilers; program control structures

Identifiers: transient variable caching; **stack** based intermediate representation; register based architectures; unoptimized compiled code; transient variable usage; optimization practices; register based machines; transient variables; **stack** based environments; transient values; **Java compilers** ; instruction reordering; **stack** manipulation operations; **stack** based IR code sequences; decompilation

Class Codes: C6150C (Compilers, interpreters and other processors); C6110J (Object-oriented programming); C6140D (High level languages); C6150N (Distributed systems software); C6120 (File organisation)

31/5/23 (Item 11 from file: 2)

DIALOG(R)File 2:INSPEC

6258020 INSPEC Abstract Number: C1999-07-6150C-007

Title: Support for garbage collection at every instruction in a Java/sup TM/ compiler

Author(s): Stichnoth, J.M.; Guei-Yuan Lueh; Cierniak, M.

Author Affiliation: Intel Corp., Santa Clara, CA, USA

Journal: SIGPLAN Notices Conference Title: SIGPLAN Not. (USA) vol.34, no.5 p.118-27

Publisher: ACM,

Publication Date: May 1999 Country of Publication: USA

CODEN: SINODQ ISSN: 0362-1340

SICI: 0362-1340(199905)34:5L.118:SGCE;1-8

Material Identity Number: S202-1999-006

Conference Title: Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation (PLDI)

Conference Sponsor: ACM SIGPLAN

Conference Date: 1-4 May 1999 Conference Location: Atlanta, GA, USA

Language: English Document Type: Conference Paper (PA); Journal Paper (JP)

Treatment: Practical (P)

Abstract: A high-performance implementation of a Java Virtual Machine requires a compiler to translate Java bytecodes into native instructions, as well as an advanced garbage collector (e.g., copying or generational). When the Java **heap** is exhausted and the garbage collector executes, the compiler must report to the garbage collector all live object references contained in physical registers and **stack** locations. Typical compilers only allow certain instructions (e.g., call instructions and backward branches) to be GC-safe; if GC happens at some other instruction, the compiler may need to advance execution to the next CC-safe point. Until now, no one has ever attempted to make every compiler-generated instruction CC-safe, due to the perception that recording this information would require too much space. This kind of support could improve the GC performance in multithreaded applications. We show how to use simple compression techniques to reduce the size of the GC map to about 20% of the generated code size, a result that is competitive with the best previously published results. In addition, we extend the work of Agesen, Detlefs, and Moss (1998), regarding the so-called "JSR Problem" (the single exception to Java's type safety property), in a way that eliminates the need for extra runtime overhead in the generated code. (11 Refs)

Subfile: C

Descriptors: Java; program compilers; **storage** management
Identifiers: **Java compiler** ; **garbage collection** ; high-performance Java Virtual Machine implementation; Java bytecode translation; native instructions; Java **heap** ; live object references; physical registers; **stack** locations; compiler-generated instruction; multithreaded applications; compression technique; runtime overhead
Class Codes: C6150C (Compilers, interpreters and other processors); C6110J (Object-oriented programming); C6120 (File organisation)

31/5/24 (Item 12 from file: 2)
DIALOG(R)File 2:INSPEC

6109682 INSPEC Abstract Number: C9901-6140D-036
Title: A type-based escape analysis **for functional languages**
Author(s): Hannan, J.
Author Affiliation: Dept. of Comput. Sci. & Eng., Pennsylvania State Univ., University Park, PA, USA
Journal: Journal of Functional Programming vol.8, pt.3 p.239-73
Publisher: Cambridge University Press,
Publication Date: May 1998 Country of Publication: UK
CODEN: JFPRES ISSN: 0956-7968
SICI: 0956-7968(199805)8:3L.239:TBEA;1-I
Material Identity Number: N848-98005
U.S. Copyright Clearance Center Code: 0956-7968/98/$12.50
Language: English Document Type: Journal Paper (JP)
Treatment: Theoretical (T)
Abstract: An important issue faced by implementers of higher-order functional programming languages is the allocation and deallocation of **storage** for variables. The possibility of variables escaping their scope during runtime makes traditional **stack** allocation inadequate. We consider the problem of detecting when variables in such languages do not escape their scope, and thus can have their bindings allocated in an efficient manner. We use an annotated type system to infer **information** about the use of variables in a higher-order, strict functional language and combine this system with a translation to an annotated language which explicitly indicates which variables do not escape, The type system uses a notion of annotated types which extends the traditional simple type system with **information** about the extent of variables. To illustrate the use of this **information** we define an operational semantics for the annotated language which supports both **stack** and environment allocation of variable bindings. Only the **stack** allocated bindings need follow the protocol for **stacks** : their extent may not exceed their scope. Environment allocated bindings can have any extent, and their allocation has no impact on the **stack** allocated ones. We prove the analysis and translation correct with respect to this operational semantics by adapting a traditional type consistency proof to our setting. We have encoded the proof into the Elf programming language and typechecked it, providing a partially machine-checked proof. (22 Refs)
Subfile: C
Descriptors: functional languages; programming language semantics; **storage** allocation; type theory
Identifiers: type-based **escape analysis** ; higher-order functional programming languages; **storage** deallocation; **storage** allocation; **stack** allocation; annotated language; type system; environment allocation; variable bindings; **stack** allocated bindings; operational semantics; type consistency proof; Elf programming language; partially machine-checked proof
Class Codes: C6140D (High level languages); C4210L (Formal languages and computational linguistics); C6120 (File organisation)

31/5/25 (Item 13 from file: 2)
DIALOG(R)File 2:INSPEC

6084624 INSPEC Abstract Number: C9812-6150C-034
Title: Efficient JavaVM just-in-time compilation
Author(s): Krall, A.
Author Affiliation: Inst. fur Computersprachen, Tech. Univ. Wien, Austria
Conference Title: Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.98EX192) p.205-12
Publisher: IEEE Comput. Soc, Los Alamitos, CA, USA
Publication Date: 1998 Country of Publication: USA xiii+435 pp.
ISBN: 0 8186 8591 3 Material Identity Number: XX98-02839
U.S. Copyright Clearance Center Code: 0 8186 8591 3/98/$10.00
Conference Title: Proceedings 1998 International Conference on Parallel Architectures and Compilation Techniques
Conference Sponsor: IFIP WG 10.3; IEEE Comput. Soc.; INRIA; ACM; TELECOM Paris
Conference Date: 12-18 Oct. 1998 Conference Location: Paris, France
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Conventional compilers are designed for producing highly optimized code without paying much attention to compile time. The design goals of Java **just - in - time compilers** are different: produce fast code at the smallest possible compile time. In this article we present a very fast algorithm for translating JavaVM byte code to high quality machine code for RISC processors. This algorithm handles combines instructions, does copy elimination and coalescing and does register allocation. It comprises three passes: basic block determination, **stack** analysis and register preallocation, final register allocation and machine code generation. This algorithm replaces an older one in the CACAO JavaVM implementation reducing the compile time by a factor of seven and producing slightly faster machine code. The speedup comes mainly from following simplifications: fixed assignment of registers at basic block boundaries, simple register allocator better exception handling better **memory** management and fine tuning the implementation. The CACAO system is currently faster than every JavaVM implementation for the Alpha processor and generates machine code for all used methods of the javac compiler and its libraries in 60 milliseconds on an Alpha workstation. (16 Refs)
Subfile: C
Descriptors: parallel programming; program compilers
Identifiers: compilers; JavaVM; RISC processors; register allocation; block determination; **stack** analysis; register preallocation; CACAO system ; JavaVM implementation; native code compilation; just-in-time compilation
Class Codes: C6150C (Compilers, interpreters and other processors); C6110P (Parallel programming)

31/5/26 (Item 14 from file: 2)
DIALOG(R)File 2:INSPEC

5978043 INSPEC Abstract Number: C9809-6140-001
Title: ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI)
Journal: SIGPLAN Notices vol.33, no.5
Publisher: ACM,
Publication Date: May 1998 Country of Publication: USA
CODEN: SINODQ ISSN: 0362-1340
Material Identity Number: S202-98007
Conference Title: ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI)
Conference Sponsor: ACM
Conference Date: 17-19 June 1998 Conference Location: Montreal, Que., Canada
Language: English Document Type: Conference Proceedings (CP); Journal Paper (JP)
Abstract: The following topics were dealt with: redundancy; register

promotion; conflict misses; array languages and array access patterns; data-flow analysis; inclusion constraint graphs; flow-intensive pointer analysis; alias analysis; integer execution; branch reordering; register allocation; **garbage collection** ; **stack** collection and pretenuring; tail recursion; parallel program optimization; Cilk-5; **run - time** code **generation** ; units; array-bound checking elimination; Java; threaded code optimization; cross-module optimization; **memory** management; data members; certifying compilers; and open reactive programs.

Subfile: C

Descriptors: program compilers; programming languages; **storage** management

Identifiers: redundancy; register promotion; arrays; program analysis; **storage** allocation; program optimization; Java; **memory** management; compilers; programming language design; programming language implementation

Class Codes: C6140 (Programming languages)

31/5/29 (Item 17 from file: 2)

DIALOG(R)File 2:INSPEC

03754974 INSPEC Abstract Number: C90070747

Title: Higher order escape analysis : **optimizing** stack **allocation in functional program implementations**

Author(s): Goldberg, B.; Park, Y.G.

Author Affiliation: Dept. of Comput. Sci., Courant Inst. of Math. Sci., New York Univ., NY, USA

Conference Title: ESOP '90. 3rd European Symposium on Programming. Proceedings p.152-60

Editor(s): Jones, N.

Publisher: Springer-Verlag, Berlin, West Germany

Publication Date: 1990 Country of Publication: West Germany ix+435 pp.

ISBN: 3 540 52592 0

Conference Sponsor: Eur. Assoc. Theor. Comput. Sci

Conference Date: 15-18 May 1990 Conference Location: Copenhagen, Denmark

Language: English Document Type: Conference Paper (PA)

Treatment: Practical (P)

Abstract: The authors present a method for optimizing the allocation of closures in **memory** . This method is based on **escape analysis** , an application of abstraction interpretation to higher order functional languages. **Escape analysis** determines, at compile time, if any arguments to a function have a greater lifetime than the function call itself. Such arguments, especially if they are closures, must be allocated in the **heap** rather than in the **stack** . In most implementations, however, **stack** allocation of closures is preferable due to the lower cost of allocation and reclamation. Therefore, the **escape analysis** is used to determine when arguments can be **stack** allocated safely. In the past, first order **escape analysis** has been used in optimizing LISP compilers, and has been described in various data-flow analysis frameworks for a language with complex types. The analysis described, being higher order, provides more accurate escape **information** , although for a very simple higher order functional language. (15 Refs)

Subfile: C

Descriptors: functional programming; high level languages; **storage** allocation; **storage** management

Identifiers: **escape analysis** ; higher order functional languages; **stack** allocated; escape **information**

Class Codes: C6120 (File organisation); C6140D (High level languages)

31/5/30 (Item 18 from file: 2)

DIALOG(R)File 2:INSPEC

03232313 INSPEC Abstract Number: B88064096, C88055918
Title: High volume production: IBM achieves flexibility
Journal: Logistics Today vol.7, no.3 p.32-3, 35
Publication Date: July 1988 Country of Publication: UK
CODEN: LOTOEM ISSN: 0262-4354
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: **IBM** required a flexible materials handling system to **move** circuit boards automatically between component placement machines and testing. The system installed at the company's Greenock site by Rapistan Lande is described. The plant is highly automated and consists of two modules of 60000 ft/sup 2/ each. Cards are **stacked** in work board holders and transported among SMT placement equipment by totestackers. These totestackers are at the centre of a work area and not only **move** the product from one machine to the next but also offer the flexibility of a work in progress **buffer** store. Linking of these totestackers and other processes is achieved by the sophisticated interface of a flexible high level loop conveyor system that stretches the length of the 100 metre module. Another high loop conveyor takes finished products in plastic tote bins from the assembly module through into the 100 metre long test module where it again performs a flexible interlinking function, **moving** product between two test totestackers and a stress testing area. (0 Refs)
Subfile: B C
Descriptors: computerised materials handling; flexible manufacturing systems; printed circuit manufacture; surface mount technology
Identifiers: high volume production; FMS; **IBM** ; flexible materials handling system; component placement; testing; Greenock; Rapistan Lande; work board holders; SMT; totestackers; work in progress **buffer** store; high level loop conveyor system; plastic tote bins
Class Codes: B0170E (Production facilities and engineering); B2210D (Printed circuit manufacture); C3320 (Materials handling); C3355 (Manufacturing processes); C7420 (Control engineering)

31/5/31 (Item 19 from file: 2)
DIALOG(R)File 2:INSPEC

03187801 INSPEC Abstract Number: B88050188, C88045325
Title: The Novix NC4016 (reduced instruction set parallel microprocessor)
Author(s): Petremann, M.
Journal: Micro Systemes no.86 p.149-54
Publication Date: May 1988 Country of Publication: France
CODEN: MSYSDT ISSN: 0183-5084
Language: French Document Type: Journal Paper (JP)
Treatment: Practical (P); Product Review (R)
Abstract: A microprocessor is described which, with only 16000 CMOS transistors, consumes sufficiently low power to be suitable for battery-operated portable systems and can execute most **elementary** instructions in a single machine cycle. The structure of the word-addressed **memory** with its data **stack** , **return** **stack** and two associated pointers is represented diagrammatically. The practical **IBM** -format pluggable NB4100 extension card combines the chip with 64000 words (128 K bytes) of random-access **memory** , supplemented by 8000 words for the two **stacks** ; up to 32 independent tasks can be processed in parallel. The 40 FORTH primitives are listed, and execution times for eight routine operations are compared with those of the 8086 and 68000 processors. Four similarly equipped development or applications cards and an autonomous development station available in France are cited. The FORTH program for the classical 'sieve of Eratosthenes' prime-number-finding language performance test is listed. (7 Refs)
Subfile: B C
Descriptors: CMOS integrated circuits; microprocessor chips; reduced instruction set computing; VLSI
Identifiers: VLSI; Novix NC4016; reduced instruction set parallel microprocessor; CMOS transistors; battery-operated portable systems; machine cycle; word-addressed **memory** ; data **stack** ; **return** **stack** ;

associated pointers; **IBM** -format pluggable NB4100 extension card; random-access **memory** ; independent tasks; FORTH primitives; development; applications cards; autonomous development station; France; FORTH program; sieve of Eratosthenes; prime-number-finding language performance test; 128 kBytes
Class Codes: B1265F (Microprocessors and microcomputers); B2570D (CMOS integrated circuits); C5130 (Microprocessor chips); C5220 (Computer architecture)
Numerical Indexing: memory size 1.31E+05 Byte

31/5/32 (Item 20 from file: 2)
DIALOG(R)File 2:INSPEC

02808874 INSPEC Abstract Number: C87011853
Title: Address space indexing using an address space index. stack
Journal: IBM Technical Disclosure Bulletin vol.29, no.3 p.980-1
Publication Date: Aug. 1986 Country of Publication: USA
CODEN: IBMTAA ISSN: 0018-8689
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: For protection reasons, it is desirable to **move** from using real keys to using indexes for address space keys on an **IBM** Series/1 processor having an extended address feature. This is done using an address space index **stack** . (0 Refs)
Subfile: C
Descriptors: security of data; **storage** allocation; virtual **storage**
Identifiers: real keys; **IBM** Series/1 processor; extended address feature; address space index **stack**
Class Codes: C5310 (Storage system design); C6120 (File organisation); C6130 (Data handling techniques)

31/5/33 (Item 21 from file: 2)
DIALOG(R)File 2:INSPEC

01134965 INSPEC Abstract Number: C78001855
Title: A controller for the dynamic compiler
Author(s): Van Bree, K.A.
Journal: Hewlett-Packard Journal vol.28, no.11 p.21-3
Publication Date: July 1977 Country of Publication: USA
CODEN: HPJOAX ISSN: 0018-1153
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); Practical (P)
Abstract: The controller for the **dynamic compiler** performs all of the tasks an interpreter for APL must perform, such as handling user input and editing, sequencing between lines of a function, calling and **returning** from user-defined functions, and handling errors. In addition, the controller handles the generation and re-execution of compiled code for APL statements. (1 Refs)
Subfile: C
Descriptors: program compilers
Identifiers: controller; **dynamic compiler** ; APL; sequencing; use input handling; error handling; function calling; function **returning**
Class Codes: C6150C (Compilers, interpreters and other processors)

31/5/34 (Item 1 from file: 233)
DIALOG(R)File 233:Internet & Personal Comp. Abs.

00487213 98BY02-017
Java's object-oriented communications -- Compared to other remote-procedure-call mechanisms, RMI offers new features, thanks to tight integration with Java

. Clip, Paul
BYTE , February 1, 1998 , v23 n2 p53-54, 2 Page(s)
ISSN: 0360-5280
Company Name: JavaSoft
Product Name: Java's Remote Method Invocation
Languages: English
Document Type: Software Reviews
Grade (of Product Reviewed): B
Geographic Location: United States
Presents a favorable review of Java's Remote Method Invocation (RMI) ($NA), created by JavaSoft. Says RMI was developed specifically for Java, and was introduced as part of the Java Development Kit 1.1. States that RMI has the flexibility and scalability to allow it to **move** object implementations across systems. Explains the RMI layered architecture (which uses TCP/IP), including how the remote-procedure-call supports **dynamic class loading** . Continues that RMI employs a distributed reference counting system to handle garbage collection. Discusses how RMI uses Java's Remote Method Protocol with three subprotocols, including one that allows bidirectional communication using a single socket. Includes two diagrams. (JC)
Descriptors: Java; Networks; Object-oriented
Identifiers: Java's Remote Method Invocation; JavaSoft

31/5/35 (Item 2 from file: 233)
DIALOG(R)File 233:Internet & Personal Comp. Abs.

00442334 96PK11-302
Microsoft splitting Java -- Native-code compiler due; plans could fragment development
Moeller, Michael; Baron, Talila; Leach, Norvin
PC WEEK , November 25, 1996 , v13 n47 p1, 101, 2 Page(s)
ISSN: 0740-1604
Company Name: Microsoft
Languages: English
Document Type: Articles, News & Columns
Geographic Location: United States
Reveals plans by Microsoft to create a Windows-specific version of the Java development language. Explains that the company plans to release a **Java compiler** which will provide faster, but Windows-only, Java executables. Warns that this strategy could result in a competition between variations of Java, rather than maintaining Java as a single language. Reports that decision by Microsoft is not a popular one, and cites several complaints with regard to the **move** . Suggests that Microsoft will be fighting a losing battle if they try to turn Java into a proprietary language. Includes one sidebar. (kgh)
Descriptors: Java; Application Development; Corporate Strategy; Product Development; Competition; Window Software
Identifiers: Microsoft

31/5/36 (Item 3 from file: 233)
DIALOG(R)File 233:Internet & Personal Comp. Abs.

00426362 96MA06-104
Java development tools move **beyond minimalism**
Somogyi, Stephan
MacWEEK , June 10, 1996 , v10 n23 p1, 31-34, 4 Page(s)
ISSN: 0892-8118
Company Name: Natural Intelligence; Symantec; Metrowerks
Product Name: Roaster; Cafe; CodeWarrior
Languages: English
Document Type: Software Review
Grade (of Product Reviewed): C; B; B
Hardware/Software Compatibility: Macintosh; Power Macintosh

Geographic Location: United States
Presents a comparison review of three Java development tools from three manufacturers for use with Macintosh systems. Reviews and rates the following: Roaster DR2 ($299), an integrated development environment (IDE) with project manager, **Java compiler**, and source-code editor from Natural Intelligence Inc., rated four out of five diamonds; Cafe DR1 ($99), a project manager and interface builder with a C++ for Power Macintosh basis, from Symantec Corp., rated four out of five diamonds; and CodeWarrior v1.0.1 ($99), a **Java compiler** and linker which integrates with the existing IDE, from Metrowerks, rated four out of five diamonds. Includes two screen displays and one scorecard. (kgh)
Descriptors: Application Development; Authoring Systems; Programming Language; Macintosh; C Programming Language; Internet; Programming Language
Identifiers: Roaster; Cafe; CodeWarrior; Natural Intelligence; Symantec; Metrowerks

31/5/38 (Item 1 from file: 94)
DIALOG(R)File 94:JICST-EPlus

03409303 JICST ACCESSION NUMBER: 97A0844334 FILE SEGMENT: JICST-E
Static and Dynamic Analysis of Java Virtual Machine.
WATANABE KENJI (1); KANEDA SHOICHI (1); OTSUYAMA KOHEI (1)
(1) Aizu Univ.
Joho Shori Gakkai Kenkyu Hokoku, 1997, VOL.97,NO.76(ARC-125), PAGE.73-78, FIG.9, TBL.4, REF.6
JOURNAL NUMBER: Z0031BAO ISSN NO: 0919-6072
UNIVERSAL DECIMAL CLASSIFICATION: 681.32 681.3.06:800.92
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: Java language has been widely used as the number of internet users has grown rapidly. Java is executed with virtual machine environment called Java Virtual Machine(JVM). Usually JVM is realized as interpreter or Just In Time(**JIT**) **compiler**, Java chip which can realize JVM with hardware is already announced. In this paper, we analyzed the behavior of Java class file using static and dynamic way and discuss about suitable configuration of Java chip. We specially focused on the **cache** which can contain operand **stack** data. (author abst.)
DESCRIPTORS: computer architecture; object-oriented language; virtual machine system; special purpose processor; semiconductor chip; interpreter; compiler; performance analysis; performance evaluation; computer program; trace(computer)
BROADER DESCRIPTORS: computer system(architecture); method; programming language; formal language; language; computer system(hardware); system; hardware; solid state circuit parts; circuit component; parts; electric apparatus and parts; chip; language processor; system program; software ; analysis; evaluation
CLASSIFICATION CODE(S): JC02010O; JD03051H

31/5/39 (Item 1 from file: 95)
DIALOG(R)File 95:TEME-Technology & Management

01178417 E98020823226
Java-Prozessor
(Java processor)
anonym
Elektronik Industrie, v29, n2, pp51-52, 1998
Document type: journal article Language: German
Record type: Abstract
ISSN: 0174-5522

ABSTRACT:
Der Mikroprozessor PSC1000 (Ineltec) ist ein 32-Bit-RISC-Prozessor ohne Pipelining, Programm- oder Daten- **Caches** , aber mit einem operandenlosen 8-Bit-Befehlssatz und **Stack** -Architektur. Dadurch wird eine optimale Implementierung der Java Virtual Machine und eines **JIT** - **Compilers** moeglich. Ausgehend vom CPU, der zehn Hauptfunktionen umfasst, wird das Ressourcen-Sharing vorgestellt. Mit dem 100-MHz-Typ des PSC1000 (externer 50-MHz-Oszillator) laesst sich ein Durchsatz von fast 100 MIPS erzielen. Ueber den Input-Output-Prozessor ist eine isosynchrone Datenuebertragung zwischen Speicher und externen Systemen sowie eine Refresherzeugung fuer DRAMs moeglich. Der MPU wird naeher anhand des Blockschaltbildes beschrieben. Ueber das **Memory** Interface lassen sich verschiedene Speicher anbinden.

DESCRIPTORS: 32 BIT MICROPROCESSORS; REDUCED INSTRUCTION SET COMPUTER; COMMAND STRUCTURE; COMPUTER ARCHITECTURE; IMPLEMENTATION; COMPILERS; CENTRAL PROCESSING UNIT; DATA SIGNALLING RATE; DATA TRANSMISSION; I O UNIT; DATA **MEMORY** ; DRAM CHIPS; BLOCK DIAGRAM; OPERATOR--DATA PROCESSING; EPROM --ERASABLE PROM; SRAM CHIPS; REGISTER-- **MEMORY** ; COMPUTER INTERFACES
IDENTIFIERS: JAVA PROGRAMM; 32-Bit-Mikroprozessor; Java-Prozessor

```
Set     Items   Description
S1       1927   (DYNAMIC? OR RUN()TIME OR JUST()"IN"()TIME OR JIT OR JAVA)-
            ()(COMPILER? OR COMPILING) OR (RUN()TIME OR DYNAMIC()CODE)(2W-
            )GENERATION
S2    1485072   REALLOCATE? OR RETURN? OR REASSIGN? OR REALLOT? OR CHANGE(-
            )ALLOCAT? OR MOVE OR MOVING OR EARMARK?
S3    4800921   OBJECT? OR ELEMENT? OR ENTITY OR ENTITIES OR PACKET? OR IN-
            FORMATION
S4      77634   STACK?
S5     783333   HEAP OR TEMPORARY()STORAGE OR GARBAGE()COLLECTION OR STORA-
            GE OR BUFFER? OR CACHE? OR MEMORY OR REPOSITORY? OR UMA
S6       1784   ESCAPE()ANALYSIS OR DYNAMIC()CLASS()LOADING OR JALAPENO
S7          1   S1 (S) S2 (S) S3 (S) S4
S8         22   S1 (S) S2 (S) S3
S9          1   S1 (S) S6
S10       212   S2 (S) S3 (S) S4 (S) S5
S11         1   S10 (S) S1
S12         1   S1 (S) S2 (S) S4 (S) S5
S13        49   S1 (S) S2
S14        34   S1 (S) S4
S15        14   S14 (S) S5
S16       704   S2 (S) S4 (S) S5
S17         1   S16 (S) S1
S18         4   S16 (S) S6
S19        36   S10 (S) (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S20        71   S16 (S) (IBM OR INTERNATIONAL()BUSINESS()MACHINE?)
S21         0   S10 (S) S6
S22        13   S2 (S) S6
S23       820   S3 (S) S6
S24         7   S4 (S) S6
S25        52   S5 (S) S6
S26         6   S23 (S) S25
S27         5   S24 (S) S25
S28        36   S19 (S) S20
S29       111   S7 OR S9 OR S11 OR S12 OR S15 OR S17 OR S18 OR S22 OR S24 -
            OR S25 OR S26 OR S27 OR S28
S30        27   S19 NOT PY>2001
S31        44   S20 NOT PD>20010302
S32        38   RD (unique items)
File 810:Business Wire 1986-1999/Feb 28
         (c) 1999 Business Wire
File 647:CMP  Computer Fulltext 1988-2004/May W3
         (c) 2004 CMP Media, LLC
File 275:Gale Group Computer DB(TM) 1983-2004/May 26
         (c) 2004 The Gale Group
File 674:Computer News Fulltext 1989-2004/May W3
         (c) 2004 IDG Communications
File 696:DIALOG Telecom. Newsletters 1995-2004/May 25
         (c) 2004 The Dialog Corp.
File 621:Gale Group New Prod.Annou.(R) 1985-2004/May 25
         (c) 2004 The Gale Group
File 636:Gale Group Newsletter DB(TM) 1987-2004/May 26
         (c) 2004 The Gale Group
```